# X-Windows
# User's Guide and Reference

**Programming Family**

**IBM**

Personal
Computer
Software

SC23-0804-0

# X-Windows
# User's Guide and Reference

**Programming Family**

**IBM**

**Personal
Computer
Software**

**First Edition (September 1987)**

Portions of the code and documentation described in this book were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

This edition applies to X-Windows, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

**International Business Machines Corporation provides this manual "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time.**

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM RT PC dealer or your IBM marketing representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

# About This Book

The IBM RT Personal Computer[1] X-Windows licensed program is a windowing system that allows you to view several programs simultaneously on a bit-mapped high resolution IBM RT PC[1] display. It also provides remote display support for RT PC[1] connected by a local area network (LAN).

This book is intended for anyone using X-Windows. It is divided into two sections. Section one provides information about starting, running, customizing, and using basic X-Windows commands.

The next section contains more detailed reference material which provides information on X-Windows library functions as well as an X-Windows Technical Reference.

The appendixes in the last part of the book contain information on installation, operation and font support.

## Before You Begin

Before you can use X-Windows on an RT PC, you must have the following software and hardware installed:

- IBM RT PC AIX Operating System[2] program, Version 2.1.2 or later
- IBM RT PC X-Windows licensed program, Version 1.1 (See Appendix A, "X-Windows Installation")
- IBM RT Personal Computer Mouse
- One or more of the following IBM RT Personal Computer displays with the appropriate adapter:
    - IBM 6153 Advanced Monochrome Graphics Display
    - IBM 6154 Advanced Color Graphics Display
    - IBM 6155 Extended Monochrome Graphics Display
    - IBM 5081 Model 16 or 5081 Model 19 with an IBM Megapel Display Adapter.

---

[1] IBM RT Personal Computer, RT PC, and RT are trademarks of the International Business Machines Corporation.

[2] AIX is a trademark of the International Business Machines Corporation.

To use X-Windows on RT PCs with the Interface Program for use with TCP/IP, refer to the *Interface Program for use with TCP/IP* book.

## How to Use This Book

This section discusses the order in which information is presented in this book, as well as the way particular kinds of information appear.

Chapter 1, "Getting Started with X-Windows," contains information on starting X-Windows and running X-Windows functions.

Chapter 2, "Customizing X-Windows," discusses how to customize X-Windows, change defaults for X-Windows commands and how to automatically login to X-Windows.

Chapter 3, "X-Windows Commands," gives a description of X-Windows commands and the options associated with them.

Chapter 4, "Programming Interface to X-Windows" introduces and describes concepts about X-Windows library functions.

Chapter 5, "X-Windows Technical Reference," contains X-Windows technical information on **xterm** HFT functions, **xterm** Datastream Support, and X Server protocol requests.

Appendix A, "X-Windows Installation," explains how to install the X-Windows licensed program from the AIX shell or Usability Services, how to install fonts, and how to install X-Windows for remote usage.

Appendix B, "Fonts," describes the font support package for the RT PC.

A Reader's Comment Form and Book Evaluation Form are provided at the back of this book. Use the Reader's Comment Form at any time to give IBM information that may improve the book. After you become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

### Fast-Path Boxes and Highlighting

In the first section, you will see boxes containing instructions you should follow to perform a task on the system. In most cases, "Additional Information" or "More Detailed Information" gives you more information about each step in the box above it. This information also may include helpful hints or optional ways of doing a step.

Throughout this reference, new terms introduced in the text are shown in **boldface italics** type. These words also are defined in the glossary. Key names are shown in **boldface** type.

Command names are shown in **boldface** type, for example, **xinit** and **xterm**. Also shown in **boldface** type are file names that the system supplies or creates (for example, **/bin**).

Names of files that have been created for examples in this book are shown in `monospace black` type (for example, `AIX Shell`). Text that you type in or that appears on your display screen is shown in `monospace color` (for example, Select `Yes` for `Shutdown`).

## Sample Programs

Some sample programs are provided with the X-Windows program. These programs demonstrate some of the ways you can use X-Windows. The sample programs are stored on the X-Windows Samples Diskette. IBM makes no representations about the suitability of these programs for any purpose. They are supplied "as is" without express or implied warranty. For more information about installing the sample programs, see Appendix A, "X-Windows Installation."

## Error Messages

Error messages are provided "as is" with X-Windows.

## Related Publications

Where necessary, this book directs you to use other reference materials. The following list shows the complete names of these publications and materials:

- *IBM RT PC User Setup Guide* provides instructions for setting up and connecting devices to the system units. This book also gives procedures for testing the setup and for installing the AIX Operating System. (Packaged with *IBM RT PC Options Installation*)

- *IBM RT PC Options Installation* provides instructions for installing optional adapters in IBM RT PC IBM 6151 and IBM RT PC 6150 and installing fixed-disk and diskette drives in IBM RT PC 6150. (Packaged with *IBM RT PC User Setup Guide*)

- *IBM RT PC Installing and Customizing the AIX Operating System* provides step-by-step instructions for installing and customizing the AIX Operating System, including how to add or delete devices from the system and how to define device characteristics. This book also explains how to create, delete, or change AIX and non-AIX minidisks.

- *IBM RT PC Using the AIX Operating System* describes using the AIX Operating System commands, working with file systems, and developing shell procedures.

- *IBM RT PC AIX Operating System Technical Reference* describes the system calls and subroutines that a C programmer uses to write programs for the AIX Operating System.

This book also includes information about the AIX file system, special files, file formats, GSL subroutines, and writing device drivers. (Available optionally)

- *IBM RT PC AIX Operating System Commands Reference* lists and describes the AIX Operating System commands.

- *IBM RT PC Usability Services Reference* supplements *IBM RT PC Usability Services Guide* by including information on using all of the Usability Services commands. (Packaged with *Usability Services Guide*)

- *IBM RT PC Interface Program for use with TCP/IP* describes the Interface Program commands for transferring data among host computers, logging into remote computers, executing commands remotely, and managing networks. This book also describes the programming interfaces to the Interface Program. (Available optionally)

- *IBM RT PC Keyboard Description and Character Reference* describes the national character and keyboard support for the 101-key, 102-key, and 106-key keyboards, including keyboard position codes, keyboard states, control code points, code sequence processing, and nonspacing character sequences. (Available optionally)

## Ordering Additional Copies of This Book

- To order additional copies of this publication (without program diskettes) from your IBM representative, use Order Number SBOF-0276.

- To order from your IBM dealer, use Part Number 92X1474.

A binder is included with each order.

For information on ordering a binder, manual, or other components separately, contact your IBM representative or your IBM dealer.

# Contents

# Chapter 1.  Getting Started with X-Windows

# CONTENTS

# About This Chapter

X-Windows is a tool designed to help enhance the usability of the overall application processing environment. This is done by providing facilities that can help you work with existing application programs (such as INed and Usability Services) as well as design and implement new applications.

X-Windows permits multiple application processes to operate within multiple windows displayed on a virtual terminal. You can manage windows directly or with application programs. You can hide windows completely or partially. You can update partially hidden windows as well as windows that are completely hidden.

Each window can have a specific character set (font) associated with it. Additionally each window can have its own keyboard mapping. This capability permits character sets available on the RT PC to be connected to a specific window. Keyboard mapping coupled with the capability to access all RT PC characters provides National Language Support (NLS).

X-Windows provides many popular window management functions, including opening, moving, resizing, or circulating a window.

X-Windows provides the capability to manage local and remote RT PC displays. Remote display management can be accomplished with other RT PCs connected through TCP/IP.

X-Windows also provides a library of C functions to interface clients with servers. End-users and application programs may both be clients of X-Windows. Through various commands and calls, end-users or application programs can acquire the services of the windowing functions.

By using the steps included in this chapter, you use the following X-Windows functions:

- Starting X-Windows
- Moving and resizing a window
- Opening a clock window
- Hiding and showing windows
- Opening AIX shell windows
- Circulating a window
- Canceling a window
- Stopping X-Windows.

# Starting X-Windows

The steps in the following box tell you how to start X-Windows. Be sure that X-Windows is installed. (For installation instructions, refer to Appendix A, "X-Windows Installation.")

---

**Starting X-Windows**

1. Log in to your system.

2. At the $ prompt, type:

   xinit

3. Press **Enter**.

4. Run programs in the AIX Shell or use the window manager menu to manipulate windows.

---

# Additional Information

1. If you do not know how to log in, refer to *Using the AIX Operating System.*

   If you want X-Windows to start each time you log in, see "Logging into X-Windows Automatically" on page 2-14.

2. Next to the prompt $, type in the command xinit. The **xinit** command does three things:

   a. Starts the X Server using the **X** command. This controls the input and output of X-Windows.
   b. Opens the X-Windows window manager menu in the upper right corner of the screen using the **rtxwm** command.
   c. Opens the initial X-Windows AIX Shell window using the **xterm** command.

   **Note:** For more information on these X-Windows commands, see Chapter 3, "X-Windows Commands."

3. After entering **xinit**, X-Windows starts and you see a display similar to this:
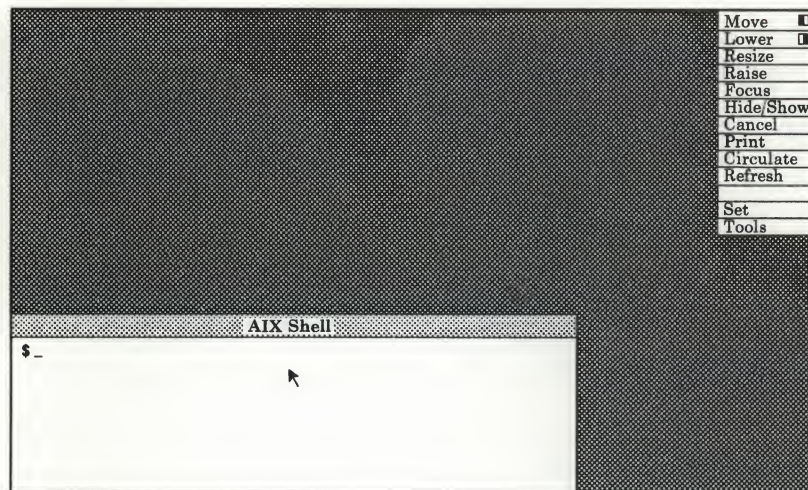


**Note:** For you to type in a window, the mouse cursor must be in that window.

An AIX Shell window functions as a terminal. The mouse cursor must be in the AIX Shell window to type into it. You can run programs just as you would on any other terminal connected to your system. For example, type `li` and press **Enter** to see the contents of your current directory.

You can use the window manager menu to perform various operations.

## Menu Selection

X-Windows provides two ways to make menu selections using the mouse. To choose an item in a menu, do one of the following:

- Use the mouse to move the cursor to the desired item, then click any button on the mouse.
- Or, press and hold a button on the mouse while you move the cursor to the desired item. Then release the button.

X-Windows highlights your selection.

For fast selection, refer to "Button/Key Selection" on page 3-20 and "Pop-up Button Selection" on page 3-20.

# Moving a Window

You can use the window manager to manipulate windows. Use **Move** to move a window. For example, you may want the AIX Shell window in a different place. When you apply **Move** to a window, a *rubber-band* outline is moved with the mouse. The rubber-band outline is the outline that appears in the window. Use the following steps to move a window:

---

**Moving a Window**

1.  Select Move from the window manager menu.
2.  Use the mouse to move the cursor to a corner of the AIX Shell window.
3.  Press and hold the same button you used in Step 1. A rubber-band outline appears.
4.  Use the mouse to move the rubber-band outline while holding the button down on the mouse.
5.  Release the button when the rubber-band outline is in the location you desire. The window is moved to fill the rubber-band outline.

---

## Additional Information

The following figure shows an AIX Shell window and the rubber-band outline created by using the **Move** item in the window manager menu:
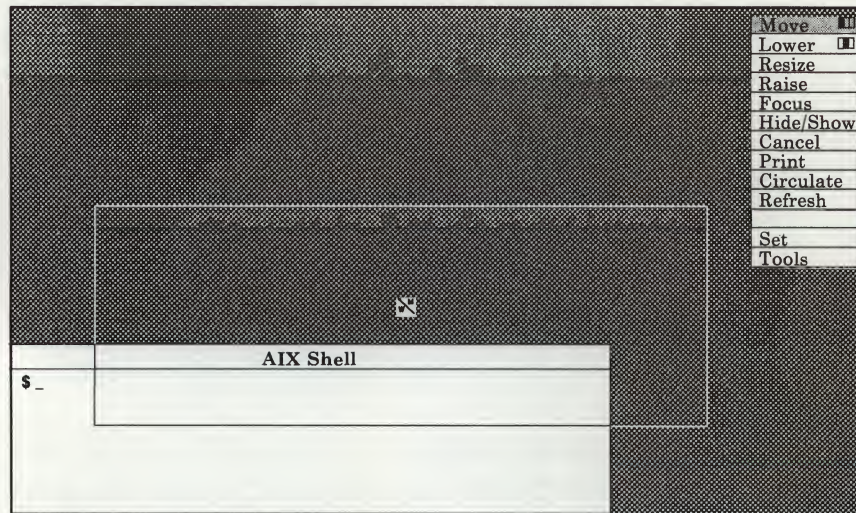
# Resizing a Window

In addition to moving a window, you can also resize it. For example, if you are using the File Manager in INed, making a window longer allows you to see more fields. Use **Resize** to resize a window by moving a corner or an edge. When you apply it to a window, a rubber-band outline of the window is displayed. Use the following steps to resize a window:

---

**Resizing a Window**

1. Select `Resize` from the window manager menu.

2. Move the cursor to any corner or edge of the window that you want to resize.

3. Press and hold the same button you used in Step 1. A rubber-band outline of the window appears, and a box appears inside the window with the screen size in it.

4. Move the rubber-band outline while holding the button down on the mouse. The numbers in the box change as you move the mouse to show the screen size in characters.

5. Release the mouse button when you have the size you want. The window is resized.

---

**Note:** You may need to restart some commands or programs after resizing a window.

The following figure shows a window with a rubber-band outline created by using the
**Resize** item in the window manager menu:

| | Move ▯▮ |
| --- | --- |
| | Lower ▯▮ |
| | Resize |
| | Raise |
| | Focus |
| | Hide/Show |
| | Cancel |
| | Print |
| | Circulate |
| | Refresh |
| | |
| | Set |
| | Tools |

060x020      **AIX Shell**

$ _
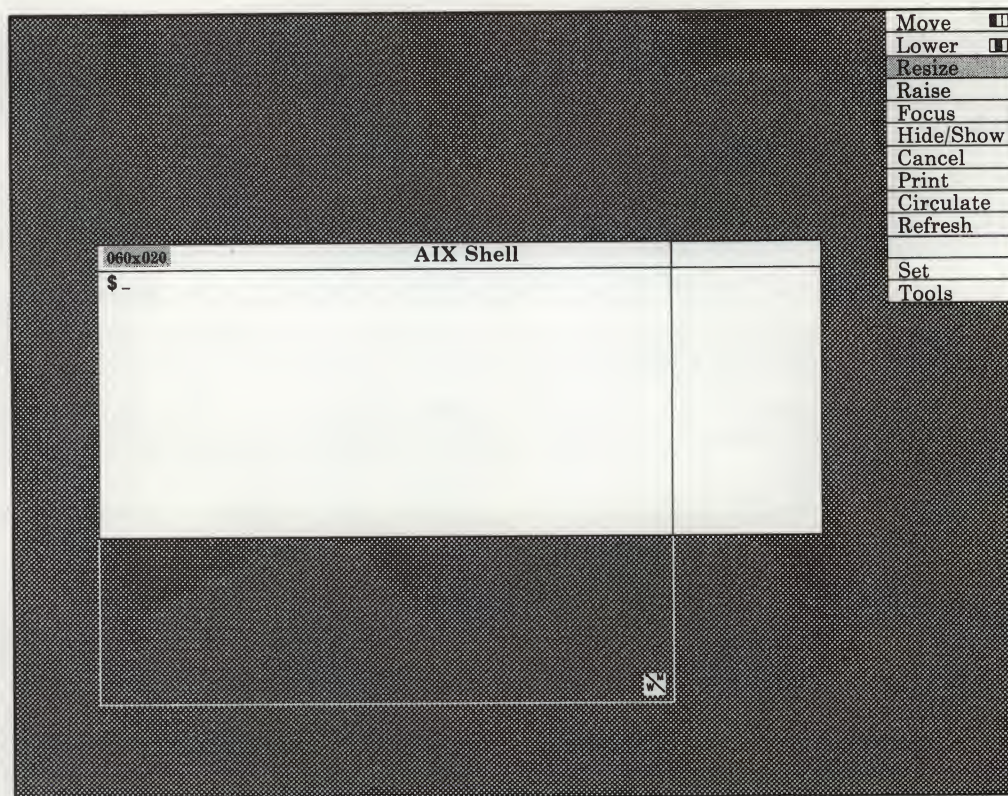
# Opening a Clock Window

There are two kinds of X-Windows clocks available on the **Tools** menu:   the Analog Clock and the Digital Clock.  Use the following steps to open the Analog Clock window:

```
┌──── Opening a Clock Window ──────────────────────────────────────┐
│                                                                  │
│  1.  Select Tools from the window manager menu.                  │
│                                                                  │
│  2.  Select Analog Clock from the Tools submenu.                 │
│                                                                  │
│  3.  The Analog Clock window is opened in the lower right corner │
│      of the display.                                             │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

The following figure shows a clock similar to the Analog Clock:
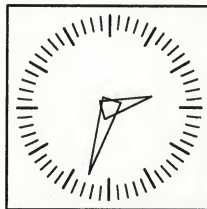
To display a digital clock, follow the same steps as those for the Analog Clock, but select Digital Clock. The Digital Clock looks similar to this on your screen:

Thu May 7 14:13:00 1987

# Hiding and Showing a Window

When you apply Hide/Show to a window, it makes the window into an icon window. When you apply Hide/Show an icon window, it makes the window reappear. Programs or commands running in a window continue running when you use Hide/Show. For example, if you are compiling a C language program in a window, you can hide it and it continues compiling. To use Hide/Show, use the following steps:

---

**Hiding and Showing a Window**

Hiding a Window

1. Select Hide/Show in the window manager menu.

2. Move the cursor into the window you want to hide.

3. Click the same button used when you made the selection. The window is represented on your display as an icon window.

Showing a Window

1. Select Hide/Show in the window manager menu.

2. Move the cursor into the icon window you want to show.

3. Click the same button used when you made your selection. The window icon is changed into a window on your display. The window appears at its previous location on your display.

---

**Note:** An icon window can be moved to any place on the display just as any other window can be moved to any place on the display.

The following is an example of an icon:



AIX #2

# Opening an AIX Shell Window

To open an AIX Shell window, use the following steps:

┌── **Opening an AIX Shell Window** ──────────────────────────┐

1.  Select `Tools` from the window manager menu.

2.  Select `AIX Shell` from the **Tools** submenu.

3.  An AIX Shell window appears.

4.  You can run programs in the AIX shell window and use the window manager to manipulate it.

└────────────────────────────────────────────────────────────┘

# Circulating Windows

`Circulate` causes the lowest window in a stack of overlapping windows to be raised. If used successfully, `Circulate` causes each window to be raised in turn. If you think of windows as being stacked on top of each other, then imagine when you circulate windows, the lowest one is raised to the top. If a window covers a large area of the display, there may be windows that you cannot see until you circulate them. To circulate among the windows, use the following steps:

┌── **Circulating Windows** ──────────────────────────────────┐

1.  Select `Circulate` in the window manager menu.

2.  The lowest window is raised to the top.

3.  Repeat the first two steps to view all the windows in order.

└────────────────────────────────────────────────────────────┘

# Canceling a Window

Cancel causes the X Server to disconnect from the selected window when the **rtxwm** is connected to an AIX X Server. The window disappears from the display. In most cases, commands or programs running in the window are also cancelled.

To cancel a window, use the following steps:

---

**Canceling a Window**

1. Select Cancel in the window manager menu.

2. Move the cursor into the window you want to cancel.

3. Click the same button used when you made your selection. The window is canceled.

---

# Stopping X-Windows

---

**Stopping X-Windows**

Press **Ctrl-Alt-Bksp** to stop X-Windows and return to the $ prompt.

---

**Note:** The $ prompt may be different on your display.

# Using Other Functions

X-Windows also provides other functions. Among these functions are the following:

- Copy and paste between terminal windows. For more information, see "The COPY, PASTE, and RE-EXECUTE Functions" on page 3-40.

- Fast window manager menu item selection. For more information, see "Pop-up Button Selection" on page 3-20.

- Change initial layout of screen. For more information, see the **xinit** command (3-35).

- Use **Set** to set various keyboard and mouse options, display the window manager horizontally, reverse video, change available colors,and set bell volume. For more information, see "Set" on page 3-23.

- Customize the window manager menu. For more information, see "Modifying the Window Manager Tools Menu" on page 2-15 and the **rtxwm** command (3-17).

# Chapter 2.  Customizing X-Windows

# CONTENTS

# About This Chapter

This chapter contains additional information that can help you customize X-Windows. It includes the following:

- Instructions for changing some defaults of X-Windows commands
- Instructions for logging in automatically to X-Windows
- Instructions for modifying the Tools menu
- A keyboard mapping chart
- Instructions for using X-Windows on a remote system
- Help with tuning the AIX Operating System for X-Windows.

# Changing X-Windows Defaults

You can set defaults such as the color, location, and size of windows by creating a file in your HOME directory. This section shows you how to set up a file and includes some sample entries. These sample entries are examples, not specifications. In some instances, you may need to use multiple keywords to fully specify a default.

## Creating the Default File

To change X-Windows defaults, first create a file named **.Xdefaults** in your HOME directory. Using this file, you can choose global defaults for X-Windows or defaults for one X-Windows command.

## Specifying Global Defaults

Specify all global defaults before any specific command defaults. The format of a global default specification is:

*keyword:value*

For example, to set the default window border to 2 pixels wide, put the following line in your **.Xdefaults** file:

```
BorderWidth:2
```

## Specifying Defaults for One Command

The format of a default specification for one command is:

*command.keyword:value*

For example, if you always want new **xterm** windows to display in reverse video, put the following line in your **.Xdefaults** file:

```
xterm.ReverseVideo:on
```

Each time you start X-Windows, windows created by the **xterm** command display in reverse video.

# Using Keywords

The next several pages describe the keywords. The descriptions include examples for each command. The examples include the commands that can use a particular keyword. In some cases, the examples include the default value supplied by IBM. In other cases, the default value may vary according to how your system is configured, and some examples may include values that are different from the default values on your system. In either case, the examples include *command.keyword:value*.

You can use this information to help you customize X-Windows. In Chapter 3, "X-Windows Commands," the commands and values are discussed in more detail.

**Note:** Some commands have flags that set options which are also specified by the use of keywords. When a command flag is used, it overrides default values set by the use of keywords.

# Keywords for X-Windows Commands

**ActiveIcon**

> **Example:**  `xterm.ActiveIcon:off`
>
> **Description:**  When a window is hidden, it is represented by an icon. Changing this value forces the window to be represented by a miniature window.

**AutoRaise**

> **Example:**  `xterm.AutoRaise:off`
>
> **Description:**  Specifies the *auto-raise* mode of **xterm**. In the auto-raise mode, a window is automatically raised when the mouse cursor enters it.

**Background**

> **Example:**  `xterm.Background:white`
>
> **Description:**  On color displays, determines the color of the background.

**BodyFont**

> **Examples:**  `rtxwm.BodyFont:Rom14.500`, `xclock.BodyFont:Rom14.500`, `xterm.BodyFont:Rom14.500`

**Description:** Specifies any fixed-width body font. These values are display dependent. For the **xterm** command in HFT emulation, the default is based on either Rom14.500 or Rom10.500, depending on the size of the display. For the **xterm** command in VT102 emulation, the default is **vtsingle**.

### BoldFont

**Example:** `xterm.BoldFont:Bld14.500`

**Description:** Specifies a bold font. This font must be the same height and width as the body font.

### Border

**Examples:** `xclock.Border:black`, `xterm.Border:black`

**Description:** On color displays, determines the color of the highlighted border.

### BorderWidth

**Examples:** `xclock.BorderWidth:1`, `xterm.BorderWidth:2`

**Description:** Specifies the width of the window border in pixels.

### C132

**Example:** `xterm.C132:off`

**Description:** Recognizes the **sm/rm** escape sequences and resizes the **xterm** window appropriately. Otherwise, the VT102 **sm/rm** escape sequences that switch between 80 and 132 column mode are ignored.

### Cursor

**Example:** `xterm.Cursor:black`

**Description:** On color displays, determines the color of the text cursor.

### DeIconifyWarp

**Example:** `xterm.DeIconifyWarp:off`

**Description:** Moves (*warps*) the mouse to the center of the window when replacing the **xterm** icon with the **xterm** window.

## Foreground

**Examples:** `xclock.Foreground:black`, `xterm.Foreground:black`

**Description:** On color displays, determines the color of the text and tick marks.

## FrameWidth

**Example:** `rtxwm.FrameWidth:5`

**Description:** Specifies the width of the border in pixels when you choose to *focus* on a window. One way to focus on a window is to choose **Focus** in the window manager menu. When you focus on a window, all keyboard input goes to that window regardless of where the mouse cursor is.

## Geometry

**Examples:** `rtxwm.Geometry:=+0+0`, `xclock.Geometry:=-0+0`, `xterm.Geometry:=80x25+0+0`

**Description:** Specifies the location or dimensions of the window. For more information about geometry, see "Geometry Specification" on page 3-4.

## Hands

**Examples:** `xclock.Hands:black`

**Description:** On color displays, determines the color of the hands in an analog clock.

## Hide

**Example:** `rtxwm.Hide:off`

**Description:** Specifies hide mode.

## Highlight

**Example:** `xclock.Highlight:black`

**Description:** On color displays, determines the color of the outline of the hands on an analog clock.

## IconBitmap

**Example:** `xterm.IconBitmap:`*filename*

**Description:** Reads the bitmap *filename* and uses the resulting bitmap as the icon.

## IconFont

**Examples:**   rtxwm.IconFont:Rom14.500, xterm.IconFont:Rom6.500

**Description:**   For the **rtxwm** command, specifies the font for the title of the window manager's window.  For the **xterm** command, specifies any fixed-width font for miniature icon windows.

## IconifyDelta

**Example:**   rtxwm.IconifyDelta:5

**Description:**   Controls where the icon is to be placed when using the **Hide** option from the window manager menu.  If this is the first time that the window has been hidden, or if the mouse is moved more than a threshold amount, the icon appears at the location on the screen where the button is released.  Otherwise, the icon reappears at its previous location. A negative value disables this effect.

**Note:**  For more information about the mouse threshold, see 3-24 and 3-29.

## IconStartup

**Example:**   xterm.IconStartup:off

**Description:**   Causes **xterm** to start by displaying an icon rather than the normal window.

## InternalBorder

**Examples:**   xclock.InternalBorder:8, xterm.InternalBorder:1

**Description:**   For the **xclock** command in analog mode, maintains an *inner border* (the distance between characters and the window's border) of eight pixels.  For the **xterm** command, maintains an inner border of one pixel.

## JumpScroll

**Example:**   xterm.JumpScroll:off

**Description:**   Enables or disables jump scroll.

## KeyCombination

**Example:**   rtxwm.KeyCombination:m

**Description:**   Specifies the selection key to be used by the window manager.

## LeftButton

**Example:** `rtxwm.LeftButton:1`

**Description:** Specifies an association between the left button and a function. For more information, see "Button/Key Selection" on page 3-20.

## LogFile

**Example:** `xterm.Logfile:/tmp/logfile`

**Description:** Specifies the file in which the log is written.

## Logging

**Example:** `xterm.Logging:off`

**Description:** Sets logging. When logging is turned on, all input from the pseudo tty is appended to the logfile.

## LogInhibit

**Example:** `xterm.LogInhibit:off`

**Description:** Prevents a user or an application program from enabling logging. This overrides any values set for **Logging**.

## MarginBell

**Example:** `xterm.MarginBell:off`

**Description:** Sets the right margin bell.

## MenuFormat

**Example:** `rtxwm.MenuFormat:h`

**Description:** Displays menu horizontally.

## MiddleButton

**Example:** `rtxwm.MiddleButton:1`

**Description:** Specifies an association between both buttons and a function. For more information, see "Button/Key Selection" on page 3-20.

**Mode**

>**Example:**   `xclock.Mode:d`
>
>**Description:**   Specifies whether the **xclock** command starts a digital or analog clock by default.

**Mouse**

>**Example:**   `xterm.Mouse:black`
>
>**Description:**   On color displays, determines the color of the mouse cursor. The default is the text cursor color.

**NMarginBell**

>**Example:**   `xterm.NMarginBell:10`
>
>**Description:**   This number is used as the right margin distance in which the margin bell rings.

**PageOverlap**

>**Example:**   `xterm.PageOverlap:1`
>
>**Description:**   A number specifies a new page overlap. In page scroll mode, a *page* is the number of lines in the the scrolling region minus the page overlap.

**PageScroll**

>**Example:**   `xterm.PageScroll:off`
>
>**Description:**   Sets the page scroll mode.

**QueueName**

>**Examples:**   `rtxwm.QueueName:lp0`
>
>**Description:**   Specifies the printer queue to use when a request is issued to print the screen.

**ReverseVideo**

>**Examples:**   `rtxwm.ReverseVideo:off`, `xclock.ReverseVideo:off`, `xterm.ReverseVideo:off`
>
>**Description:**   Reverses the foreground and background color.

## ReverseWrap

**Example:** `xterm.ReverseWrap:off`

**Description:** Setsreverse-wraparound mode, which allows the cursor to wrap from the leftmost column to the rightmost column of the previous line.

## RightButton

**Example:** `rtxwm.RightButton:p`

**Description:** Specifies an association between the right button and a function. For more information, see "Button/Key Selection" on page 3-20.

## SaveLines

**Example:** `xterm.SaveLines:64`

**Description:** When lines are scrolled off the top of a window, they can be saved. This number specifies the maximum number of lines to save.

## ScrollBar

**Example:** `xterm.ScrollBar:off`

**Description:** Specifies if the scroll bar is to be displayed during startup.

## ScrollInput

**Example:** `xterm.ScrollInput:on`

**Description:** Disables repositioning on input. When using the scroll bar to review previous lines of text, the window would otherwise by repositioned automatically at the bottom of the scroll region when input has arrived.

## ScrollKey

**Example:** `xterm.ScrollKey:off`

**Description:** Determines if the window is repositioned automatically in the normal position at the bottom of the scroll region when a key is pressed while using the scroll bar to review previous lines of text.

## SizeFont

**Example:** `rtxwm.SizeFont:Rom14.500`

**Description:** Specifies any fixed-width font as the default size font.

**StatusLine**

> **Example:** `xterm.StatusLine:off`
>
> **Description:** Determines if the status line displays on startup.

**StatusNormal**

> **Example:** `xterm.StatusNormal:off`
>
> **Description:** Determines if the status line is in normal video (the status line is still enclosed in a box). By default, the status line is in reverse-video (relative to the rest of the window).

**TextUnderIcon**

> **Example:** `xterm.TextUnderIcon:off`
>
> **Description:** Determines if text is to the right of the icon. Normally in the icon, the window name is under the bitmap.

**TitleBar**

> **Example:** `xterm.TitleBar:on`
>
> **Description:** Enables or disables the title bar from being displayed on startup.

**TitleFont**

> **Example:** `xterm.TitleFont:Rom14.500`
>
> **Description:** Specifies the font in the title bar.

**Update**

> **Example:** `xclock.Update:60`
>
> **Description:** Specifies the frequency in seconds with which **xclock** updates its display.

**VisualBell**

> **Example:** `xterm.VisualBell:off`
>
> **Description:** Sets the visual bell mode, which flashes the window on receipt of a CTRL-G.

**Warp**

    **Example:**   `xterm.Warp:off`

    **Description:**   Determines if the mouse cursor is automatically warped to the center of a newly created **xterm** window.

# Logging into X-Windows Automatically

You can run the **xinit** command and start X-Windows each time you log in to the system.

Use the **adduser** command, and change the **Program** field to `xinit -L`. The default login shell is **/bin/sh**. For more information on the **adduser** command, see *AIX Operating System Commands Reference*. For more information on the **xinit** command, see "**xinit**" on page 3-35.

The **xinit** command is a script shell file that you can modify to run other other commands, like **xclock**. Although you can modify the **xinit** command to change the default locations for the **xterm** command and the **rtxwm** command, you can change these default values along with others by using the **.Xdefaults** file in you HOME directory. For more information on using the **.Xdefaults** file, see "Changing X-Windows Defaults" on page 2-4.

**Note:** If you modify the **xinit** command, make sure that the exec **/usr/lpp/X/bin/xterm** is the last command issued. Any other command might become the controlling terminal process.

# Modifying the Window Manager Tools Menu

You can modify the menu that appears when you select Tools from the window manager menu. To make the necessary changes, you must log in as **su** or have superuser authority.

The values for the Tools menu are in the **/usr/lpp/X/defaults/Xtools.txt** file. Two examples are shown below:

1. Copy the **/usr/lpp/X/defaults/Xtools.txt** file into your HOME directory. You can then modify this file without affecting other X-Windows users on your system.

2. One of the lines in the **/usr/lpp/X/defaults/Xtools.txt** file. contains the following information:

   ```
   ¦   ¦   ¦        ¦xclock =-0-0 -a &        ¦Analog Clock   ¦
   ```

   You can change any of the values of this line and modify the way the analog clock looks when you choose Analog Clock from the Tools menu. For example, if you change the =-0-0 to =+0-0, the analog clock starts in the lower left corner instead of the lower right corner.

3. You can also add programs to the Tools menu. For example, you can add an option to start a new X Server from the Tools menu by inserting the following line in the **/usr/lpp/X/defaults/Xtools.txt** file:

   ```
   ¦   ¦   ¦        ¦xopen xinit   ¦X      ¦Run another X Server
   ```

   By using the new item this adds to the Tools menu, you can start another X Server without leaving X-Windows.

# Keyboard Mapping

X-Windows allows each window to have its own keyboard mapping.

The following keyboard source files are delivered with the X-Windows licensed program:

- keymap.gr — Austrian/German
- keymap.be — Belgian
- keymap.cf — Canadian (French)
- keymap.de — Danish
- keymap.uk — English (UK)
- keymap.us — English (US)
- keymap.sw — Finnish/Swedish
- keymap.fr — French (AZERTY)
- keymap.it — Italian
- keymap.ja — Japanese English
- keymap.no — Norwegian
- keymap.po — Portuguese
- keymap.sp — Spanish
- keymap.sf — Swiss (French)
- keymap.sg — Swiss (German)
- keymap.vt — VT102

At installation time, the language menu allows you to select any or all of these languages. The VT102 keyboard mapping is always installed. These files are installed into the directory **/usr/lpp/X/defaults**. The first language selected during installation is the one that is compiled into binary form.

The following examples show commands issued to perform a specific keyboard mapping task. They all assume that:

- The default mapping is English (US)
- The appropriate source maps are installed
- The commands are issued from an X-Windows window.

**Example 1 — Building a VT102 keyboard map**

```
cd /usr/lpp/X/defaults
mkdir vt
keycomp < keymap.vt > vt/.Xkeymap
```

**Example 2 — Run xterm with VT102**

```
XDIR=/usr/lpp/X/defaults/vt
export XDIR
xterm -v
```

**Example 3 — Build the French and Spanish maps**

```
cd /usr/lpp/X/defaults
mkdir fr
mkdir sp
keycomp < keymap.fr > fr/.Xkeymap
keycomp < keymap.sp > sp/.Xkeymap
```

**Example 4 — Run a French, Spanish, and English (US) X-Windows window**

```
XDIR=/usr/lpp/X/defaults/fr
export XDIR
xterm
XDIR=/usr/lpp/X/defaults/sp
export XDIR
xterm
```

*Keyboard Description and Character Reference* gives you the detailed mappings of the keyboards for each national language.

# Tuning System Parameters for X-Windows

X-Windows makes extensive use of the AIX operating system and its resources. You may be able to improve the performance of X-Windows by tuning system parameters. The following pages provide information about tuning the following areas:

- ptys
- processes
- X Server malloc space.

For additional information, see *Managing the AIX Operating System*.

## ptys

Each window opened by the **xterm** command uses one pty. You have two ways of defining the limits on the number of ptys:

- The number of ptys configured into the kernel
- The number of pty device nodes in **/dev**.

### Kernel pty Customization

By default, the kernel is configured with 16 ptys. You can change this number and rebuild the kernel to adjust the number of possible ptys. Use the following steps to change the number of possible ptys:

1. Edit the **/etc/master** file.
2. Modify the **ptybuffers** attribute in the **sysparms** stanza.
3. Modify the **maxminor** attribute in the **uptc** and **upts** stanzas.

Each pty uses some kernel memory. Other system parameters (such as number of processes and charlists) should be tuned to reflect any additional ptys. Each pty implies at least two processes in use: one for the control and one for the slave.

As you increase the number of ptys, you should also increase the number of charlists. Each charlist (or cblock) has space for 64 characters. Try to have a minimum of three or four charlists for each pty to be in use at the same time. For ptys that are heavily used, increasing the number of charlists may improve performance.

**Note:** It is possible to run out of charlists and hang the system.

If an X Server is hidden by another virtual terminal, there are processes (such as **xterm**) writing to that server, and if the sockets to the server fill up, the ptys may fill up on the

slave-to-master path and use all the charlists. The blockage can be resolved by showing the X Server, allowing its display backlog to disappear and freeing charlists.

To avoid running out of charlists, provide enough charlists so that **xterm** slave processes can block on output without using up all the free charlists. This means you should provide approximately five additional charlists (about 300 characters) for each pty.

The charlists are defined by the **charlists** attribute in the **sysparms** stanza of the **/etc/master** file.

### System pty Customization

Each device is declared in a stanza of **/etc/system**. Use the **devices** command to add devices to the system. The **devices** command adds devices to the configuration files and makes a special device node in **/dev**. Many programs other than X-Windows use ptys. Most of the other programs require the use of a **getty** that supports login. You may have more ptys in the kernel than you have defined by **devices**.

The maximum number of ptys is 256 (the maximum number of minor devices per major device). Ordinarily, you should not use **devices** to create more than 64 ptys.

## Processes

The maximum number of processes is defined by the **procs** attribute in the **sysparms** stanza of the **/etc/master** file. Increase this parameter if you are using X-Windows intensively.

To change the number of processes:

1. Edit the **/etc/master** file.

2. Change the **procs** attribute in the **sysparms** stanza.

3. Rebuild and install the kernel.

Once you increase the number of processes to about 100, you need to increase some additional parameters. These parameters are:

charlists          number of clists for tty subsystem

filetab            number of files the system can have open at once

inodetab           number of inodes the system can have open at once.

The **filetab** and **inodetab** should be the same.

## X Server malloc Space

The X Server does a **malloc** to get space for the various objects it creates and manipulates. If the the **ulimit** is too low, the server may run out of space.

Use the **sh** command to increase the **ulimit**. For more information, see *AIX Operating System Commands Reference*.

# Using X-Windows on a Remote System

You can work on another RT PC through an X-Windows client program on your RT PC. You use X-Windows on a remote computer system in the same way you use it on your own system. However, you must be able to access and log in to the remote system. For more information, see "Installation Requirements for Remote Usage" on page A-23.

Starting X-Windows on the remote system after logging on to that system allows you to work with programs and files stored on both your system and the remote system at the same time, through different windows. This enables you, for example, to display a file stored on your own system and another file stored on the remote system side-by-side through different windows. You can also edit a file or run a program stored on one system through one window while you run another program on another system through another window.

In summary, X-Windows allows you to have immediate access to both your own computer system and to the processing power, programs and files stored on a remote system.

# A Sample Remote X-Windows Session

This section explains the steps you go through to use X-Windows on a remote system. Steps are listed in the box. The detailed explanations that follow the box contain examples of what you can enter on your system to perform each step.

---

**Steps in Remote X-Windows Usage**

1. Start an AIX Shell window on your display.

2. Enable a particular remote system to use your display.

3. Log in to the remote system and start an X-Windows client program on the remote system to display on your local screen.

4. Work just as you work on your own system.

5. End the client program.

---

## More Detailed Information

The examples in the following explanations assume that:

- Two RT PC systems are attached to one another through a communications link.
- The program TCP/IP manages the communications between the two systems and is installed and running correctly on both machines.
- X-Windows is installed and running correctly on both machines.
- The hostname of your RT PC (the *local system*) is frank.
- The hostname of the RT PC attached remotely to your RT PC (the *remote system*) is richard.
- You are logged on to and working at frank.
- You know how to log in to a remote computer system.
- You want to edit a file stored on system richard using system frank.
- There is a single X Server running on system frank.

To use X-Windows to edit a file on a remote system, you perform the following steps:

1. To start an AIX Shell client program on your display, first select Tools from the menu. The **Tools** submenu appears on your display. Then select AIX Shell from the **Tools** submenu. A window with the window name AIX Shell appears on your display.

   **Note:** In this example, the hostname of your RT PC is frank and the AIX Shell window is the first X-Windows client program that you open on frank. Therefore, the full default name of the display in which the AIX Shell window is running is frank:0. frank is the hostname and 0 is the display number.

2. To enable the remote system richard to use your display, you enter the X-Windows **xhost** command. First you move the mouse cursor into the AIX Shell window. Then you enable the remote system richard for X-Windows by entering:

   xhost richard

   The execution of **xhost** enables the specified remote system only until you terminate X-Windows. However, you can eliminate the need to run **xhost** to enable a remote system by enabling the system by default in a file called **/etc/X?.hosts** (? is the display number).

   For example, the display frank:0 can be accessed by systems defined in the file **/etc/X0.hosts** on the system with a hostname of frank. In both the display name and the file name, 0 indicates the number of the display that the remote system is allowed to access using X-Windows.

There must be a separate **/etc/X***n***.hosts** file on the local system for each display that a remote system will access through X-Windows.

For more information on the **xhost** command, see "**xhost**" on page 3-33.

3. Log in to the remote system from the AIX Shell window on your system and open an X-Windows client program that runs on the remote system but displays on your system.

For instance, if TCP/IP is the communications program managing the data link between your system frank and the remote system richard, you can enter the following **rexec** command to log in to richard and open a window that runs on richard and appears on your display (attached to frank):

```
rexec richard xterm frank:0 -n RICHARD
```

The parts of the TCP/IP **rexec** command define the following:

| rexec | The TCP/IP command that sends a specified command to run on a specified remote system. **rexec** initiates a login process on the remote system that must complete successfully before the command is executed. |
|---|---|
| richard | The name of the remote system on which the command is to be run. |
| xterm | The X-Windows command that is to be run on the remote system. In this case, **xterm** opens a new X-Windows client program on richard. |
| frank:0 | A parameter of the **xterm** command that indicates the full name of the display where the new window is to appear. In this case, the new window running on richard appears on your display which is physically attached to frank. The hostname frank and the display number 0 must be separated by a : (colon). |
| -n RICHARD | A flag of the **xterm** command that indicates the window name to be used for the new window. |

For more information on the **rexec** command, see *Interface Program for use with TCP/IP*. For more information on the **xterm** command, see "**xterm**" on page 3-39.

The **xterm** command causes a rubber-band window to appear on your display (frank:0) after you complete the login initiated by the **rexec** command. You can press and hold down a mouse button to move the rubber-band window. When you release the mouse button, the window border becomes a solid line and the window name RICHARD appears at the top of the new window.

**Note:** Although the work you perform in the new X-Windows client program is primarily processed by the remote system richard, your current hostname is not changed. Your current hostname is still frank (the name of your system) and the

RICHARD window is the second window that you open from that current host. Therefore, the full default name of the display that the remote window RICHARD uses is frank:0.

4.  At this point, for example, you can start an editor and edit a file stored on the remote system richard through the remote window named RICHARD.

    In general, through a window running on a remote system, you can run programs and access files that your login user ID on richard has permission to run and access. For example, you can use the programs and files stored on the remote machine richard through the remote X-Windows client program RICHARD. At the same time, through another window, you can use any program and file stored on your local system frank that your local user ID has permission to use.

5.  When you complete your work on the remote system richard, you enter **Ctrl-D** to shut the remote window RICHARD. This action also logs you off of the remote system.

**Note:** TCP/IP may not be required to run remote client programs. For example, a system administrator might write a program to put up messages in an X window. The system administrator can open such a message window on a remote system if the following conditions are met:

•  The remote system name is known.

•  The remote system allows access.

# Chapter 3.  X-Windows Commands

# CONTENTS

# About This Chapter

This chapter discusses general command information and describes the X-Windows commands in alphabetical order.  Each command description includes the following sections:

- A **Purpose** section one or more sentences long.

- A **Syntax** diagram that shows the required, optional, and default command syntax.

- A **Description** section that gives a detailed explanation of command usage.

- A **Flag** section that discusses each flag that can be specified with the command. Where appropriate, the default setting of the flag is given.

Explanations of file structures and command submenus are included where relevant.

Also, an opening section on syntax diagrams discusses how to interpret the parts of the command syntax diagrams.

# General Information

The following sections discuss topics that are applicable to several different X-Windows commands.

## Command Defaults

A user can customize X-Windows by copying the file **/usr/lpp/X/defaults/Xdefaults** into the home directory (**$HOME**) as **.Xdefaults** and customizing the values defined in the file. The format of the file is:

*command.keyword:string*

If you omit *command*, the specified default value is used for all appropriate X-Windows commands. Global defaults must appear in the file before any specific program defaults.

Each command has keywords that correlate to the command arguments. For more information about keywords and default values, see "Changing X-Windows Defaults" on page 2-4 and the discussions of specific commands.

## Geometry Specification

Most commands accept a ***geometry specification*** allowing creation and placement of windows on the screen. A geometry specification is written in the following format:
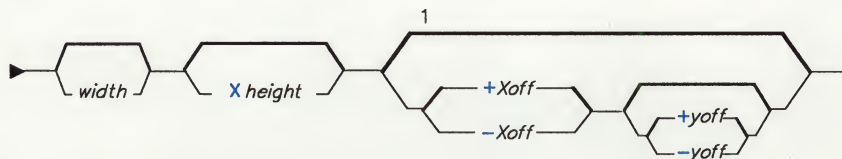


[1] Mouse must be used to create window.

You specify the *width* and *height* as the number of characters for text programs and usually as pixels for graphics programs. The offsets *Xoff* and *Yoff* are specified as pixels.

If you do not specify an offset, you must use a mouse to create a window. If you specify a size and an offset, a window is automatically created when the program begins.

*Xoff* and *Yoff* specify distances from a corner of the screen to the nearest corner of the window in the following way:

| + *Xoff* + *Yoff* | Upper left to upper left. |
|---|---|
| − *Xoff* + *Yoff* | Upper right to upper right. |
| + *Xoff* − *Yoff* | Lower left to lower left. |
| − *Xoff* − *Yoff* | Lower right to lower right. |

## Keyboard Specification

You can change the standard keyboard layout or the default values of the keymap and function keys. Some programs search for and use the **.Xkeymap** file in the home directory of the user for setting up key and function key input resolution.

The **.Xkeymap** file is produced by the **keycomp** program. **.Xkeymap** is the file used to translate keystrokes into character strings.

Many programs perform the translation process by calling the library routine **XLookupMapping**. **XLookupMapping** searches for the keymap table in the following order:

1. **$XDIR/.Xkeymap** (program directory)

2. **$HOME/.Xkeymap** (home directory)

3. **/usr/lpp/X/defaults/.Xkeymap** (system directory)

4. A built-in table that provides 7-bit ASCII character mapping.

Depending on what combinations of the **Shift**, **Lock**, **Ctrl**, **Alt**, and **Alt Graphic** keys you use, each key can have up to 32 different interpretations or *bindings*. (The **Alt Graphic** keys are only on non-US keyboards.) With US English keyboard mapping, for example, pressing **A** produces A (a capital *A*) when **Shift** or **Lock** is down, octal 001 when **Ctrl** is down, and a (a small *a*) when no other key is down.

For more information on keyboard mapping, see "Keyboard Mapping" on page 2-16. For more information on producing a customized **.Xkeymap** file, see "**keycomp**" on page 3-12.

# Color Specification

Many programs allow you to specify colors for things such as the text or the screen background. A color specification can be given as either a color name (such as **blue**) or as a string of three hexadecimal values with each value specifying the intensity of the red, green, or blue color components.

The color names are defined in the **/usr/lib/X/rgb/rgb.txt** file. The following is a list of some of the colors defined in the file:

- Black
- Blue
- Cyan
- Green
- Navy
- Red
- Tan
- White
- Yellow

The hexadecimal values must be given in one of the following formats:

*#RGB*

*#RRGGBB*

*#RRRGGGBBB*

*#RRRRGGGGBBBB*

In this table, R, G, and B represent single hexadecimal digits (upper- or lowercase). When fewer than 16 bits each are specified, they represent the most significant bits of the value. For more information about **XParseColor**, see "XParseColor" on page 4-76.

**Note:** When using one of these values as part of a **sh** (shell) command, enclose the value in double quotation marks. Normally, # indicates a comment in a shell script.

## Display Specification

When you first run the **xinit** command, the **$DISPLAY** environment variable is set to the following string:

*name:number*

The contents of this variable specify the display used by programs running with X-Windows:

- The *name* is usually the hostname of a particular system.

- The *number* is used to specify a specific X Server on the named system.

Some commands accept a display specification. This causes the command to run on the named system and to display on the numbered X Server on that system.

# Syntax Diagrams

Before each command discussion, you will find a syntax diagram that shows you how to enter that command correctly on the command line. These diagrams show:

- Which flags can be entered on the command line
- Which flags must take parameters
- Which flags have optional parameters
- Default values of flags and parameters, if any
- Which flags can and cannot be entered together
- Where you must enter flags or parameters and where you have a choice
- Where you can repeat flag and parameter sequences.

The following discussion explains how to interpret the syntax diagrams. It begins with an example diagram that shows most of the conventions used in diagrams. Each part of the diagram is labeled and explained. Following the example are sample diagrams.

Diagram items that must be entered literally on the command line are in **bold**. These items include the command name, all flags, and literal characters. Variable items are in *italics*. These items include parameters that follow flags, and parameters that the command reads, such as *files* and *directories*. If an item has a default value, it is shown in the normal font and the path is shown in bold. You do not enter on the command line any item shown in the normal font on a bold path.

The following diagram illustrates the conventions used in the syntax diagrams:
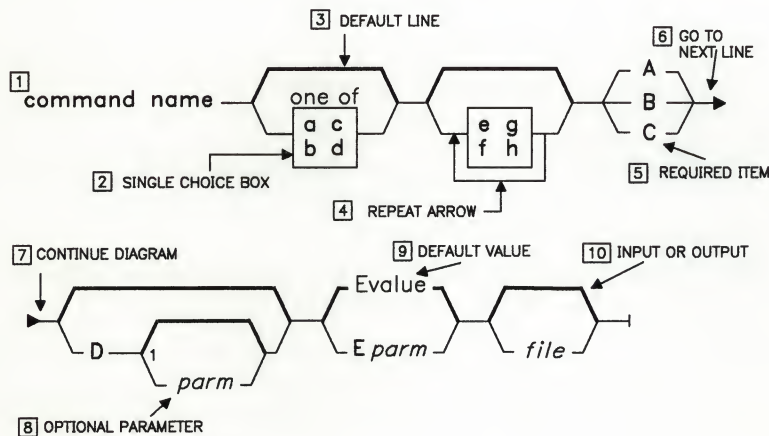


¹ Do not put a blank between these items.

You interpret the diagram as follows:

**1 command name**  The first item in the diagram is the name of the command you want to invoke. It is in bold, so it must be entered exactly as it appears in the diagram.

After the command name, the path branches into two paths. You can follow either path.

**2 SINGLE CHOICE BOX**  If you follow the lower path, you encounter a box with the words one of over it. You can choose only one item from this box.

**3 DEFAULT LINE**  If you follow the upper path, you bypass the single choice box, and enter nothing. The bold line around the box is a default line, which means that you do not have to enter anything from that part of the diagram. Exceptions are usually explained under "Description." One important exception, the blank default line around input and output files, is explained in item **10**.

**4 REPEAT ARROW**  When you follow a path that takes you to a box with an arrow around it, you must choose at least one item from the box. Then you can either follow the arrow back around and continue to choose items from it, or you can continue along the path. When following the arrow around just the box

(rather than an arrow that includes several branches in the diagram), do not choose the same item more than once.

**5** REQUIRED ITEM

Following the branch with the repeat arrow is a branch with three choices and no default line around them. This means that you must choose one of **A**, **B**, or **C**.

**6** GO TO NEXT LINE

If a diagram is too long to fit on one line, this character tells you to go to the next line of the diagram to continue entering your command line. Remember, the diagram does not end until you reach the vertical mark.

**7** CONTINUE DIAGRAM

This character shows you where to continue with the diagram after it breaks on the previous line.

**8** OPTIONAL PARAMETER

If a flag can, but does not have to, take a parameter, the path branches after the flag to show this. If you cannot enter a space between the flag and parameter, you are told in a footnote.

**9** DEFAULT VALUE

Often, a command has default values or actions that it will follow if you do not enter a specific item. If the default is not something you can enter on the command line, it is not indicated in the diagram. However, it is discussed under "Flags."

**Note:** Default values are included in the diagram for your information. Do not enter them on the command line.

**10** INPUT OR OUTPUT

A command that can read either standard input or input files has an empty default line around the file parameter. If the command can write its output to either a file or to standard output, it is also shown with a default line around the output file parameter. If a command can read only from standard input, input is not shown in the diagram, and standard input is assumed. If a command writes only to standard output, this is also assumed and output is not included in the diagram. When you must supply a file name for input or output, the file parameter is included in the diagram without a default line around it.

Following are examples of how to enter this command based on this syntax diagram:

```
command name A
command name C
command name a B
command name d B
command name e A
command name e g f A
command name C D
command name C D8
command name A E7
command name B myfile
command name a e g B D3 E6 myfile
command name d f e h C D myfile
```

**Note:** Although the diagram implies that the order of the flags is important, it is usually not. When the order of the flags is important, it is indicated in the diagram, under "Flags," or in both places. With this in mind, an additional example of how to enter this command is:

```
command name E9 a D g A h f myfile
```

# keycomp

## Purpose

Reads in a textual description of the keyboard and produces a binary keymap file.

## Syntax

keycomp —< *infile* —> *outfile* ⊣

## Description

The **keycomp** command (*keycomp* is an abbreviation for *keymap compiler*) reads in a textual description of the keyboard and produces a binary keymap file. The keymap file is used to translate keystrokes into character strings. For more information on the keymap file, see "Keyboard Specification" on page 3-5.

The **keycomp** command supports the full range of HFT keyboard mapping, including the **Alt Graphic** shift state.

You can use **keycomp** to define *diacritical* keys (dead keys). The code point combinations that produce the actual diacritical characters are pre-defined and cannot be changed using **keycomp**. The pre-defined combinations are listed in **XLookupMapping**.

Five different states are supported in the base keymap files. Additional states are either mapped to single states or defined as **UNBOUND** (return nothing) for the RT PC keymap files.

### Keycomp Source File

The input file to **keycomp** consists of one or more lines, each beginning with an octal or decimal number designating an X-Windows keyboard code. Items follow the key code, each representing the binding for a particular combination of the **Ctrl**, **Alt**, **Shift**, **Lock**, **Alt Graphic** keys. Items on the line are separated by a space.

If only one item is present on a line, it represents the binding for this key code regardless of the position of the shift keys. The first 16 states are required in the source file. If more

than 16 but fewer than 32 states are provided, the last state is extended to all the missing states up to state 32.

The bindings of items are made in the order defined below:

| #1 | Base state; no **Ctrl**, **Alt**, **Shift**, **Lock**, or **Alt Graphic** down |
|---|---|
| #2 | **Lock** down |
| #3 | **Shift** down |
| #4 | **Shift** and **Lock** down |
| #5 | **Alt** down |
| #6 | **Alt** and **Lock** down |
| #7 | **Alt** and **Shift** down |
| #8 | **Alt**, **Shift**, and **Lock** down |
| #9 | **Ctrl** down |
| #10 | **Ctrl** and **Lock** down |
| #11 | **Ctrl** and **Shift** down |
| #12 | **Ctrl**, **Shift**, and **Lock** down |
| #13 | **Ctrl** and **Alt** down |
| #14 | **Ctrl**, **Alt**, and **Lock** down |
| #15 | **Ctrl**, **Alt**, and **Shift** down |
| #16 | **Ctrl**, **Alt**, **Shift**, and **Lock** down |
| #17 | **Alt Graphic** down |
| #18 | **Alt Graphic** and **Lock** down |
| #19 | **Alt Graphic** and **Shift** down |
| #20 | **Alt Graphic**, **Shift**, and **Lock** down |
| #21 | **Alt Graphic** and **Alt** down |
| #22 | **Alt Graphic**, **Alt**, and **Lock** down |
| #23 | **Alt Graphic**, **Alt**, and **Shift** down |
| #24 | **Alt Graphic**, **Alt**, **Shift**, and **Lock** down |
| #25 | **Alt Graphic** and **Ctrl** down |
| #26 | **Alt Graphic**, **Ctrl**, and **Lock** down |
| #27 | **Alt Graphic**, **Ctrl**, and **Shift** down |

| #28 | **Alt Graphic**, **Ctrl**, **Shift**, and **Lock** down |
| #29 | **Alt Graphic**, **Ctrl**, and **Alt** down |
| #30 | **Alt Graphic**, **Ctrl**, **Alt**, and **Lock** down |
| #31 | **Alt Graphic**, **Ctrl**, **Alt**, and **Shift** down |
| #32 | **Alt Graphic**, **Ctrl**, **Alt**, **Shift**, and **Lock** down |

## Keycomp Source File Items

Each item should be one of the following:

- An octal or decimal number, indicating a character code.

- A C character literal surrounded by single quotes. Escape sequences (such as \252) are allowed.

- A C string literal surrounded by double quotes. Standard C escape sequences are allowed within the string.

- The letter **U**, indicating no binding. If there is no binding, **XLookupMapping** returns an empty string for this key combination.

- The string format "**D**$nn$." to define a key position as a diacritical key. There are 15 pre-defined diacritical keys. **XLookupMapping** combines a specified diacritical key with the following key pressed to determine the actual code point to be returned. The code point returned is based on the pre-defined diacritical lookup table. Strings "**D01**" through "**D15**" are not allowed for **keycomp**.

A comma can, but need not, follow each item. A space or tab must separate the items whether or not a comma follows each item.

Blank lines are ignored, as are lines beginning with the # character (except control statements). All text between a # character and the following line is ignored unless the # is part of a string enclosed in single or double quotes. This allows you to place comments at the end of a line that contains only a single item.

The **keycomp** command can identify function key strings supported by the RT and compress these within the keymap file. The set of function key strings is defined in the "Set Keyboard Map" section of *AIX Operating System Technical Reference*.

The source must specify the exact string to be returned.

See the files **/usr/include/X/Xkeyboard.h** and **/usr/include/X/Xkeymap.h** for a list of key codes and key names of function keys.

## Keycomp Source File Control Statements

The following control statements are recognized by **keycomp**:

1. **#S Control Statement**

   Lines starting with **#S** in the first column define which states are defined within the **keycomp** table. This allows the states not being used to be compressed out of the keymap file. If this line is not specified, it is assumed that all states are built into the table. All states must be coded in the source file.

   The states not included in **#S** are **UNBOUND** and return nothing unless remapped to another state (see the **#M** control statement).

   The **keycomp** object file provides a **state_mapping_table** to map keyboard state flags to indexes into the table. The **state_mapping_table** maps the state detail of a **KeyPressed** event from an X Server to an index within the keymap table.

   Following **#S** is a series of numbers representing the states defined in the table. The states provided are built into the table in the order in which they are defined.

   For example, the **Ctrl** key is normally mapped to index **9** in the keymap file. With the following definition:

   #S 1 2 3 5 9 17

   the **Ctrl** key is mapped to index **5** because state **#9** is the fifth state in the **#S** statement.

2. **#M Control Statement**

   Lines starting with **#M** in the first column define mapping of states to a index within the keymap table. This allows specification of a state hierarchy as defined for the RT PC and allows mapping of multiple states to a single state. For example, the **#M** statement enables **Ctrl-Shift** keys to be mapped to **Ctrl** keys.

   The format of a **#M** line is:

   **#M** *STATE s1 s2 ... sn*

   where states *s1, s2, ... sn* are mapped to state *STATE*. *STATE* is a base state depending on the **#S** specifications.

   The **#M** line must follow all **#S** lines. Multiple **#M** lines can be specified but must be specified after the **#S** statement.

For example, the following line:

```
#M 9 11 12
```

maps the **Ctrl-Shift** and **Ctrl-Shift-Lock** states to **Ctrl**.

## Flags

| | |
|---|---|
| *< infile* | Specifies a source file to be compiled by **keycomp**. |
| *> outfile* | Specifies the name of the keymap file to be created. |

## Files

To be compatible with *Keyboard Description and Character Reference*, RT PC keyboard files supplied with X-Windows contain the following control statements:

```
#S 1 2 3 4 5 9 17
#M 5 6 7 8 21 22 23 24
#M 9 10 11 12 13 14 15 16 25 26 27 28 29 30 31 32
#M 17 18 19 20
```

# rtxwm

## Purpose

Provides window manager functions.

## Syntax



```
rtxwm ──┬─────────────┬──┬──────────┬──┬──────────────────┬──┬───────────┬──────
        └─ name: number ─┘  └─ geometry ─┘  ├─ -fi font ─┤        ├─ t= f ────┤
                                            ├─ -fn font ─┤        └─ selectkey ┘
                                            ├─ -fs font ─┤
                                            ├─ -h ───────┤
                                            ├─ -H ───────┤
                                            ├─ -x num ───┤
                                            ├─ -• num ───┤
                                            └─ -r ───────┘
```
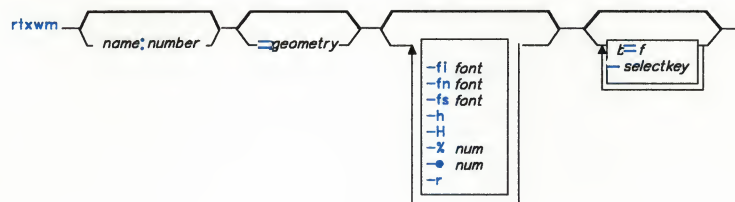
## Description

The window manager allows you to manipulate the windows on the screen.  X-Windows:

- Implements overlapping windows
- Allows windows to be moved, hidden, and resized
- Allows the order of the windows in a stack of overlapping windows to be manipulated
- Allows the keyboard focus to be attached to a window
- Allows the invocation of commands from a window
- Allows various display options to be set.

The window manager normally takes control of the screen at various times to assure that the screen image remains correct while performing window manager operations.  When this happens, requests from other applications are temporarily suspended until the window manager finishes the operation.

# Flags

| | |
|---|---|
| $b = f$ | Specifies an association between a button ($b$) and a function ($f$). $b$ can be one of the following: |

l  left
r  right
m  both (middle).

l indicates the left mouse button, **r** indicates the right mouse button, and **m** indicates the both mouse button. When using a three-button mouse, specify **m** by pressing the middle button.

$f$ can be one of the following functions:

c  circulate
C  Cancel
f  focus
h  hide/show
l  lower
m  move
p  pop
P  print
r  raise
R  Refresh
S  Set
T  Tools
z  resize.

**pop** specifies the button that is used to pop up the command menu at the position of the mouse cursor.

| | |
|---|---|
| − **fi** *font* | Specifies an icon font for use in place of the default icon font. Any fixed-width font can be used. The default for this flag is **Rom14.500** for a large display and **Rom10.500** for a small display. |
| − **fn** *font* | Specifies a font for use in place of the default font. Any fixed-width font can be used. The default for this flag is **Rom14.500** for a large display and **Rom10.500** for a small display. |
| − **fs** *font* | Specifies a particular size font for use instead of the default size font. Any fixed-width font can be used. The default for this flag is **Rom14.500** for a large display and **Rom10.500** for a small display. |
| = *geometry* | Specifies the location of the **rtxwm** window. The default for this flag is $= -0 + 0$. Values for width and height, if entered, are not used. For more information, refer to "Geometry Specification" on page 3-4. |
| − **h** | Displays the menu horizontally. The default value for this flag is vertical. |

| | |
|---|---|
| −**H** | Specifies hide mode. The default is off. |
| *name:number* | Identifies the hostname and display number where **rtxwm** is to run. Normally, the hostname and display number are obtained from the environment variable **DISPLAY**. For more information, refer to "Display Specification" on page 3-7. |
| % *num* | Controls where the icon is to be placed when hiding a window. The default is **5**. |
| | If the mouse is moved more than a threshold amount or if this is the first time the window has been hidden, the icon appears at the location of the mouse cursor when the button is released. Otherwise, the icon reappears at its previous location. A negative value disables this effect. |
| @ *num* | Specifies the width in pixels of the border when a window is focused. The default value for this flag is **5**. |
| −**q** | Specifies the printer queue to use when a request is issued to print the screen. |
| −**r** | Enables reverse video. |
| − *selectkey* | Specifies the selection key used in combination with the mouse button to select menu items automatically. *selectkey* is one of the following: |

- **a** signifying left **Alt** (**Alt**)
- **c** signifying **Ctrl**
- **g** signifying right **Alt** (**Alt Graphic**)
- **l** signifying **Lockshift**
- **m** signifying left **Alt** (**Meta**)
- **s** signifying **Shift**
- **n** signifying none.

The default for this flag is **a**.

## Menu Modes

The window manager has two modes of operation:

- Normal
- Hidden.

For more information about the hidden mode, see "Button/Key Selection" on page 3-20.

The default mode is normal. The default pop-up button is the right button. The default selection key is the **Alt** key.

In the normal mode, the command menu is always visible. The menu window's home position is the upper right corner of the screen. To perform an action, you click any mouse button in the appropriate menu box and click the same button in the window you wish to select.

To activate hidden mode, you use the −**H** option. In the hidden mode, the menu's home position is hidden until it is popped up. The command menu pops up when the pop-up button is pressed. At least one button must be defined to cause the command menu to pop up. Whenever the pop-up button is clicked while the appropriate combination of **Ctrl**, **Alt**, and **Shift** keys are pressed, or any time a button not bound to the **Focus** function is clicked in the background, the menu appears beneath the cursor. You can then use the menu as defined for the pop-up button.

## Selection Methods

Selection within the menu can be done with one of the following methods:

- Moving the mouse cursor to the window manager menu and selecting a menu item with any button.

- Pressing the pop-up button as previously described to view the window manager menu, then release the button at a menu item.

- Pressing button and key combinations for automatic selection. This mechanism allows a key, in combination with a mouse button, to automatically select a menu item and immediately apply the function to a window. Automatic selection is applied to the window containing the mouse cursor.

### *Pop-up Button Selection*

Pressing the pop-up button (by default with **Alt** down) moves the command menu with the previously selected item or the central one beneath the mouse cursor. The menu remains at that location until an item is selected or until the mouse cursor is moved out of the menu. By default, the pop-up button is the right button, but it can be defined to be any button.

When a command is selected:

- The menu item remains selected until the command is applied to a window.

- The menu is returned to its previous state and location if **rtxwm** is in normal mode. If the menu is in hidden mode, the menu is removed from the display.

If the mouse cursor is moved out of the menu, nothing is selected. This is useful if you decide not to select an item once the menu is activated.

### *Button/Key Selection*

X-Windows reserves certain button/key combinations and interprets them as operations on existing windows. Button/key selection can be used in place of the default mouse button and menu selection method to automatically select and run an operation.

The key combination can be specified in the command line with some subset of the options:

- **a** signifying left **Alt** (**Alt**)
- **c** signifying **Ctrl**
- **g** signifying right **Alt** (**Alt Graphic**)
- **l** signifying **Lockshift**
- **m** signifying left **Alt** (**Meta**)
- **s** signifying **Shift**
- **n** signifying none.

**Note:** **Alt** refers to the left **Alt** key.

For example, if you specify the options −**ca**, the **Ctrl** and **Alt** keys must be down at the time a mouse button is pressed. The option −**n** means that no keys need to be held down. This is not recommended because it means that application programs never receive unshifted mouse clicks.

If no combination is specified in the command line, **Alt** is assumed.

## Window Manager Command Menu

The window manager displays a menu of commands that you can use to manipulate windows on the display. By default, the menu is displayed vertically in the upper right corner of the display.

On your screen, the window manager command menu looks similar to the following:

```
Move       ▣
Lower      ▣
Resize
Raise
Focus
Hide/Show
Cancel
Print
Circulate
Refresh

Set
Tools
```

Commands are arranged according to frequency of use, with the top-left item being the most-used and the bottom-right item being the least-used. All but the last two items act on windows.

You use the menu by selecting a item within the menu, then applying the command to a window. Once you select a menu item, **rtxwm** controls the mouse until the command is applied to a window. You can deselect an item on the menu by clicking a different button.

For example, to hide a window, you can use the following steps:

1. Move the mouse cursor to `Hide/Show` in the menu and select it. The item `Hide/Show` is highlighted in the menu.

2. Move the mouse cursor to the window to be hidden and press the same button.

3. The window is hidden. A window icon is displayed and `Hide/Show` is unhighlighted.

The commands in the window manager command menu provide the following functions:

`Move`    Moves a window. When you select a window, you can use the mouse to move an outline of the window. When you release the button, the window is moved.

`Lower`    Pushes the window you select to the bottom of any stack of overlapping windows.

`Resize`    Resizes a window by moving a corner or an edge. When you apply it to a window, an outline of the window is displayed. Moving the mouse cursor changes the size of the outline, leaving the opposite corner or other edges fixed. The corner or edge that moves depends on the location of the mouse cursor when the button is pressed.

The window is divided into a logical grid of eight rectangles. If the mouse cursor is in one of the four corner rectangles, the corner closest to the mouse cursor is moved; otherwise, the closest edge is moved. When the button is released, the window is resized.

`Focus`    Attaches the keyboard to a window. Keyboard input goes to that window even when the mouse cursor is outside the window. It also raises the focused window. Focusing the background detaches the keyboard from any window by attaching it to the background window. When no window is focused, the keyboard input goes to the window that contains the mouse cursor. The focused window is highlighted by a partial frame.

`Hide/Show`    Makes a window into an icon or an icon into a window. When applied to an icon, `Hide/Show` makes the original window reappear at its former position on the screen.

If a window has not provided an icon window, the window manager creates its own icon window and places the title of the window in it. In this case, the mouse movement and editing functions discussed in this section are valid.

If the mouse is moved more than a threshold amount or if this is the first time the window is being hidden, the icon appears at the location on the screen where the button is released. Otherwise, the icon reappears at its previous location.

The threshold amount can be changed with the %*num* flag. Giving a negative value disables this effect.

The icon name can be edited. Pressing the **Delete** or the **Backspace** key deletes the last character of the icon name, pressing **Ctrl-U** deletes the entire

name, and pressing other character keys appends the characters to the current name.

Cancel    Causes the X Server to disconnect from the selected window. The window is taken away. Applications usually terminate when disconnected from the X Server.

**Note:** This only works if **rtxwm** is connected to an AIX Operating System X Server.

Raise     Raises a window to the top of any stack of overlapping windows.

Print     Prints the contents of a window on the printer. The printer device name is obtained form the environment variable XPRINTDEV (for example XPRINTDEV="-device 3812"). Printer devices are supported as shown below:

**3812**  IBM 3812 Pageprinter
**5201**  IBM 5201 Quietwriter Model 2
**5202**  IBM 5202 Quitewriter III.

Circulate Causes the lowest window in the stack of overlapping windows to be raised. Successive applications reveal each window in turn.

Refresh   Clears the display and forces each application to redraw its contents.

Set       See "Set."

Tools     See "Tools" on page 3-25.

Both Set and Tools display a submenu below or above the mouse cursor location, depending on the space available. The submenu remains visible until a selection is made or the mouse cursor is moved out of it.

## Set

Selecting Set from the window manager command menu displays a submenu through which you can set various display options. Some of the options are toggle buttons that can be set either on or off. If an option is marked with a + (plus sign), the option is set to on.

The following table lists the options on the **Set** menu:

| | |
|---|---|
| Autorepeat | Enables or disables key repeat while a key is pressed. |
| Shift Key | Enables or disables the shift lock mode. |
| Hide Menu | Causes **rtxwm** to hide the command menu until it is activated. Once a command is complete, the command menu is hidden again. |
| Horizontal Menu | Enables or disables the horizontal display of menu items. |
| Reverse Video | Reverses foreground and background colors in the window manager menu. |
| Assign Button | Displays a copy of the command menu and enables the association of a mouse button with a menu item. Clicking a button while the mouse cursor is on a menu item associates the button with the item. |
| Click | Sets the keyboard click to either off or to a volume level from 1 through 8. A menu with the current volume is displayed. Pressing the right button increases the value and pressing the left button decreasesthe value. Pressing both buttons sets the volume. |
| Foreground Color | Displays a menu of available colors from which you can select a foreground color. |
| Background Color | Displays a menu of available colors from which you can select a background color. |
| Bell | Sets the bell volume. A menu with the current volume is displayed. Pressing the right button increases the value and pressing the left button decreases the value. Pressing both buttons sets the volume. |
| Mouse | Sets the acceleration and threshold for the mouse. A menu for each value is displayed in sequence. Pressing the right button increases the value and pressing the left button decreases the value. Pressing both buttons sets the value. |
| Screen | Sets the length of time in minutes before the server clears the screen. A menu with the default value is displayed. Pressing the right button increases the value and pressing the left button decreases the value. Pressing both buttons sets the time. |

## Tools

Selecting Tools displays a menu of application program names that can be invoked within X-Windows. Using this menu, you can select and start programs within X-Windows. The **Tools** menu supports the invocation of three classes of programs:

| | |
|---|---|
| X-Windows applications | Application programs written directly to the X library and invoked by their command names. |
| Emulation applications | KSR application programs that are supported by the **xterm** HFT emulation function; invoked with the **xterm − e app** command. |
| Full-screen applications | Programs that write directly to the display adapter card and run in monitor mode; invoked with the **xopen app** command. |

## The Tools Menu Controller

The file **/usr/lpp/X/defaults/X.txt** controls what is displayed in the command menu. The format of a line in this file is:

*function_context*ǁ*function_name*ǀ*description*

**rtxwm** uses the *function_context* field to invoke a function or open another pop-up text file and the *function_name* field to build the command menu. The *description* is a comment field.

You can modify **X.txt** by editing its contents with a text editor or by using the **Menu Update** command. This command is similar to the **Tools Update** menu in Usability Services. To use **Menu Update**, Usability Services must be installed on the system. For more information about Usability Services, see *Usability Services Reference*.

A file named **Xtools.txt** for the **Tools** pop-up is added within **X.txt**. The default **Xtools.txt** contains the AIX shell and the analog and digital clock applications.

## The Tools Menu File

The file **Xtools.txt** contains information on application programs accessible through the **Tools** window. The format of a line in this file is:

ǀǀǀ*exec_program*ǀ*command_name*ǀ*description*

**rtxwm** uses the *exec_program* field to invoke the application program. The *exec_program* field allows a command string to be supplied rather than just a command name. This allows the customization of commands. The *command_name* field is used to build the **Tools** pop-up. The *description* is a comment field.

For example, the command:

```
xterm =80x24 -fn Rom14.500 -e dos
```

specifies that DOS Services be invoked in a new window with the **Rom14.500** font. This must be done in order to support invocation of full-screen applications.

## Default Keywords

The default keywords for use with the **rtxwm** command are:

**BodyFont**
**FrameWidth**
**Geometry**
**Hide**
**IconFont**
**IconifyDelta**
**KeyCombination**
**LeftButton**
**MenuFormat**
**MiddleButton**
**QueueName**
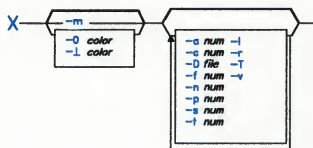**ReverseVideo**
**RightButton**
**SizeFont**

For more information about the use of these keywords, see "Changing X-Windows Defaults" on page 2-4.

# X

## Purpose

Starts the X Server.

## Syntax



## Description

The *X Server* is a display server that runs on computers with bitmapped terminals. The X Server distributes user input to and accepts output requests from programs located either on the host system or on systems connected to it through a network.

Unless you specify otherwise, only programs running on the host system can interact with the display. To allow another system to use your display, you must define that system to a specific X Server with the **xhost** command. For more information on the **xhost** command, see "**xhost**" on page 3-33.

After the X Server is initialized, it sends `unix:?` IBM X-Windows 1.1 to standard output, where *?* denotes the display number. This string is used by the **xinit** command to set the default **DISPLAY** environment variable.

The X Server and all windows opened from it can be terminated by pressing **Ctrl-Alt-Backspace**. Remote windows usually display an error message concerning a broken connection before they terminate.

The X Server logs messages in **/tmp/.X?.msgs**, where *?* is the display number.

## Flags

The following flags have default values supplied with the program:

**−a** *num*   Specifies the acceleration. The default value is **4** pixels. The **acceleration** is a multiplier for mouse movement. For example, specifying **4** causes the cursor to move four times as fast as the mouse. The specified value must be a positive value greater than 0.

**−c** *num*   Specifies the key click volume. The default value is **6**. The following values are supported:

| | |
|---|---|
| **0** | off |
| **1, 2, 3** | low |
| **-1, 4, 5, 6** | medium |
| **7, 8** | high |

**−0** *color*   Specifies a background color for the display. The default value depends on the display.

**−1** *color*   Specifies a foreground color for the display. The default value depends on the display.

**−D** *file*   Specifies the full pathname of the color definition data base file to use. The default is **/usr/lpp/X/rgb/rgb**.

**−f** *num*   Specifies the beep volume. The default value is **6**. The supported values are the same as those supported for the **−c** *num* flag.

**−l**   Disables shift lock. The default is shift lock enabled.

**−m**   Specifies the use of monochrome display characteristics.

**−n** *num*   Specifies the connection number. Valid values for *num* are **0** to **255**. The default is the next available number. *num* is used by programs to communicate with a specific X Server. For example, the command:

```
X -n 18
```

specifies that communication to the activated X Server takes place by **unix:18** or by *hostname*:**18**.

**−p** *num*   Specifies the number of minutes to elapse between connection checks. The default is **60** minutes. A specified value must be a positive number greater than 0.

**−r**   Disables auto repeat. The default is auto repeat enabled.

**−s** *num*   Specifies the number of minutes to wait until making the display blank. The default is **10** minutes. A specified value must be a number greater than 0.

−t *num*      Specifies the mouse threshold. The default value is **2** pixels. Acceleration takes effect only if the mouse is moved more than the mouse threshold in one time interval and only applies to the amount beyond the threshold.

−**T**      Disables the **Ctrl-Alt-Backspace** key sequence that, by default, terminates the X Server and all windows opened from it.

−**v**      Replaces the display with the current background color, after the amount of time specified by the −**s** flag, By default, if the −**v** flag is not specified, the entire display is painted with the background tile after the amount of time specified by −**s**. On color displays, random foreground and background colors are also used.

# xclock

## Purpose

Continuously displays the current time of day.

## Syntax
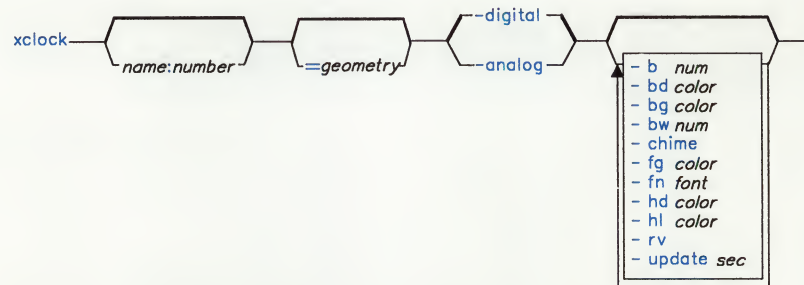


## Description

The **xclock** command gets the time from the system clock. This time is displayed and updated by X-Windows in the form of either a digital or an analog clock.

## Flags

| | |
|---|---|
| **−analog** | Sets analog display mode. Draws a conventional 12-hour clock face with ticks for each minute and stroke marks on each hour. The default is digital mode. |
| **−b** *num* | Specifies the width in pixels of padding white space between the window border and anything **xclock** displays. The default is **10** in digital mode and **2** in analog mode. |
| **−bd** *color* | Specifies the border color on color displays. The default is black. |

| | |
|---|---|
| **−bg** *color* | Specifies the color of the background on color displays. The default for this flag is white. |
| **−bw** *num* | Specifies the width in pixels of the border. The default value for this flag is **1**. |
| **−chime** | Specifies the sounding of a chime every 60 minutes on the hour. The default is off. |
| **−digital** | Sets digital display mode. Displays date and time in digital form. |
| **−fg** *color* | Determines the color of the text and tick marks on color displays. The default for this flag is black. |
| **−fn** *font* | Specifies a font for use instead of the default font. Any fixed-width font can be used. The default for this flag is **Rom14.500**. |
| **=** *geometry* | Specifies the location and dimensions of the window. The default setting is **= −0−0**. For more information on *geometry*, refer to "Geometry Specification" on page 3-4. |
| **−hd** *color* | Specifies the color of the hands in analog mode on color displays. The default for this flag is black. |
| **−hl** *color* | Specifies the highlight color. For example, the outline of the hands of the analog clock can be highlighted with this color. The default is black. |
| *name:number* | Identifies the hostname and display number where the clock is to run. Normally the hostname and display number are found in the environment variable **DISPLAY**. Refer to "Display Specification" on page 3-7. |
| **−rv** | Reverses foreground and background colors. |
| **−update** *sec* | Specifies the frequency in seconds with which **xclock** updates its display. If the **xclock** window is obscured and then exposed, **xclock** overrides this and redisplays immediately. The default update frequency is **60** seconds. The specification of an update frequency greater than 30 seconds disables the display of the second hand in analog mode. |

# xclock

## Default Keywords

The default keywords for use with the **xclock** command are:

**Background**
**BodyFont**
**Border**
**BorderWidth**
**Foreground**
**Geometry**
**Hands**
**Highlight**
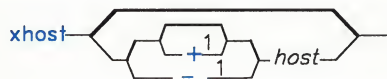**Mode**
**ReverseVideo**

For more information about the use of these keywords, see "Changing X-Windows Defaults" on page 2-4.

# xhost

## Purpose

Controls who can have access to X-Windows on the current host machine.

## Syntax

```
xhost ──┬──────────────────────────┬──┤
        │      ┌──── +¹ ────┐       │
        └──────┤           ├─host─┘
               └──── −¹ ────┘
```

```
¹
Do not put a blank after these items.
```

## Description

The **xhost** command adds and deletes hosts on the list of machines from which X-Windows accepts connections. It is only sufficient for a work station (single-user) environment.

This command must be executed on the machine to which the display is connected. You can remove a host from the access list by using the −*host* option. Do not remove the current host from the access list. If you do, you must log off the system before making any corrections.

Entering **xhost** with no arguments shows the names of the hosts currently allowed to access your X-Windows display.

To enable a remote host by default, the host can be defined in a file named **/etc/X?.hosts** (? is the display number to which you enable access).

For example, the display **frank:0** can be accessed by systems defined in the file **/etc/X0.hosts** on a system that uses the default hostname of **frank**. In both the display name and the file name, **0** indicates the number of the display that the defined remote systems are allowed to access through X-Windows.

# xhost

## Flags

| | |
|---|---|
| *host* | Specifies a host node ID number and adds the host to the X-Windows access list. |
| + *host* | Specifies a host node ID number and adds the host to the X-Windows access list. (Same as the *host* option; the + is optional.) |
| − *host* | Specifies a host node ID number and deletes a host from the X-Windows access list. |

# xinit

## Purpose

Starts an X Server with a single command.

## Syntax

xinit ─┬─────┬─┬──────────────┬─┬─────────────────┬─
       └ −L ┘ └ X_options ┘ └ xterm_options ┘

## Description

The **xinit** command starts the X Server, an **xterm** window, and an **rtxwm** window manager. It can be entered from the AIX command line or as a user's login command specified in the **/etc/passwd** file. If used as a login command in **/etc/passwd**, the user is automatically logged into X-Windows.

**xinit** performs the following operations:

- Executes the user's profile, depending on the − **L** option
- Starts an X Server on the current display
- Sets up the **DISPLAY** environment variable
- Starts the **rtxwm** command
- Starts the **xterm** command.

**xinit** uses the **SHELL** environment variable to start the command within **xterm**.

If **xinit** is the invoked login program, the termination of the initial window set up by the window manager automatically terminates all other windows opened from that window.

The **xinit** command is a shell script that can be customized to include any commands you wish and to open as many windows as you wish.

If **xinit** is invoked from **/dev/console**, a new virtual terminal is opened and an X Server is started on the new virtual terminal.

## xinit

## Flags

−L     Specifies that **xinit** be used as the login program and that the profile of the user (**$HOME/.profile**) be read and executed. Otherwise the profile is assumed to be set up.

*X_options*     Specifies any valid X options that do not conflict with *xterm_options*.

*xterm_options*     Specifies any one of the three valid **xterm** options:

- = *geometry*
- − **e**
- − **n**

These options are passed to the **xterm** command that opens the initial window. This allows the customization of the location, size and contents of the initial window.

The default for = *geometry* is = **80x12** + **0** + **0**. You use the − **e** option to execute an initial command within the login window. For example, the following line in **/etc/passwd** starts X-Windows with the DOS Services as the login shell:

/usr/bin/xinit -L -e /usr/bin/dos

# xopen

## Purpose

Opens a full-screen window (virtual terminal) and monitors it.

## Syntax

xopen ──┬─────────────────┬──┬──────────┬──┬──────┬──┬─────────────┬── cmd ──┤
        └─ name: number ──┘  └─ -ib file ┘  └─ -m ┘  └─ -n name ────┘

## Description

The **xopen** command monitors the full-screen window as follows:

- A virtual terminal is opened for the full-screen application.
- An icon is created in the X-Windows display for the full-screen application.
- The Show function of the virtual terminal activates the full-screen application.
- When the full-screen application ends, the icon is removed from the X-Windows display.

## Flags

| | |
|---|---|
| *cmd* | Specifies a command to be executed within the full screen window. Any number of valid command arguments can also be entered. |
| **-ib** *file* | Specifies the name of a bitmap icon file to be used instead of the default icon bitmap. This file, assumed to be in bitmap format, is read and the resulting bitmap is used as the icon. |
| **-m** | Turns off monitoring of the virtual terminal. The icon is not displayed in the window and no monitor process is created. |
| **-n** *name* | Provides a window name. If no name is provided, the command name is used as the window name. |

## xopen

*name:number*     Identifies the hostname and the display number of the display
          where **xopen** is to run.

# xterm

## Purpose

Initializes an X-Windows terminal emulator.

## Syntax
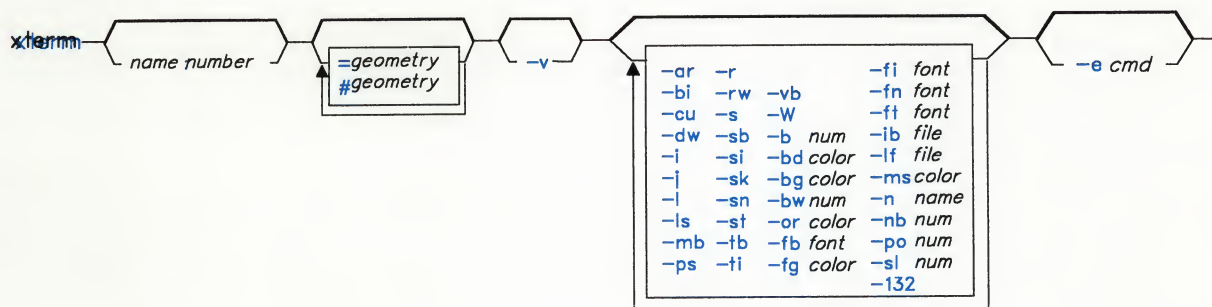


## Description

The **xterm** command provides a standard terminal type for programs that do not interact directly with X-Windows. It can emulate either an HFT terminal or a VT102 terminal. The default is HFT emulation. The VT102 mode is activated by the −**v** flag.

Depending on the emulation mode, the environment variable **TERM** may be one of the following:

- **ibm6153** (monochrome display) for HFT mode
- **ibm6154** (color display) for HFT mode
- **vt100** for VT102 mode.

In order to make **xterm** device-independent, only **ibm6153** or **ibm6154** are used. **xterm** supports the display of up to 16 colors at a time.

The **xterm** terminal supports escape sequences that perform terminal functions such as cursor control, moving and deleting lines, and **xterm** private functions.

Many of the special **xterm** features (like the scrollbar and logging) can be modified under program control through a set of private **xterm** escape sequences. You can also use escape sequences to change the title in the title bar and to specify a new logging file name.

For more information on these escape sequences and the supported data streams, see "xterm Datastream Support" on page 5-8 and *AIX Operating System Technical Reference*.

There are four different areas in the **xterm** window:

- Title bar
- Scroll bar
- Status line
- Terminal window.

By default, only the title bar and the terminal window are initially displayed.

The terminal window is the area provided for the terminal emulation. When you create a window, a pseudo terminal is allocated and a command (usually a shell) is started.

The **xterm** command automatically highlights the window border, text cursor, and title bar when the mouse cursor enters the window (selected) and unhighlights them when the mouse cursor leaves the window (unselected). If the window is the *focus window*, the window is highlighted regardless of the location of the mouse cursor.

In addition, if input comes in while an **xterm** window is hidden and the icon of the window is a static bitmap, a box is drawn around the icon title. **xterm** also may display a miniature version of the terminal window when it is hidden. This allows a user to monitor a window while it is hidden. The default is to display the static bitmap when the window is hidden.

The environment variable **WINDOWID** is set to the X-Windows ID number of the **xterm** window.

## The COPY, PASTE, and RE-EXECUTE Functions

Once you create a window, **xterm** allows you to save text and restore it within the same or other windows by using COPY, PASTE, and RE-EXECUTE button functions. These text functions are enabled when holding down the **Shift** key and are available in both HFT and VT102 emulation. The selected text is highlighted while the button is pressed.

The COPY, PASTE, and RE-EXECUTE button functions perform as described below:

| COPY | Pressing **Shift** and **both** mouse buttons at once (or the **middle** button on a three-button mouse) saves text into the cut buffer. You move the cursor to the beginning of the text and hold the button down while moving the cursor to the end of the text you want to save. The text is highlighted as you move the cursor. When you release the button, the selected text is saved in the global cut buffer. COPY does a text cut, not a box cut. |
|---|---|

| PASTE | Pressing **Shift** and the **right** mouse button types the text from the cut buffer into the window that contains the mouse cursor, inserting it as keyboard input. |
|---|---|
| RE-EXECUTE | Pressing **Shift** and the **left** mouse button takes the text from the cursor (at button release) through the end of line (including the new line), saves it in the global cut buffer and immediately retypes the line, inserting it as keyboard input. The selected text is highlighted. Moving the mouse cursor off of the initial line cancels the selection. If there is no text beyond the initial cursor point, **xterm** sounds the bell, indicating an error. |

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. The cut buffer is globally shared among different applications. The terminal emulator treats the cut buffer like a text file, in that the text is delimited by new lines.

## Menu Usage

The **xterm** command has three different menus:

- Options
- Modes
- Scrollbar

Each menu pops up under the correct combinations of key and button presses. Each menu contains various modes that can be toggled. Most of the menu items can also be altered by the use of command options. A + (plus sign) appears next to a mode that is currently active. Selecting one of these modes toggles its state. Some items of the menus are command entries; selecting one of these performs the indicated function.

The Options menu pops up when **Ctrl** and the **left** mouse button are pressed in a window. The menu contains items that apply to all emulation modes. This menu can also be activated by pressing the **left** mouse button while the mouse cursor is in the title bar.

The Modes menu sets various modes for each emulation mode. The menu is activated by pressing **Ctrl** and **both** mouse buttons at once (or the **middle** button on a three-button mouse) while the mouse cursor is in the window. The soft reset entry resets scroll regions, a function that can be useful when a program leaves the scroll regions set incorrectly. The full reset entry clears the screen, resets tabs to every eight columns, and resets the terminal modes (such as wrap and smooth scroll) to their states after **xterm** initially finished processing the command line options. This menu can also be activated by pressing **both** mouse buttons at once (or the **middle** button on a three-button mouse) while the mouse cursor is in the title bar.

The Scrollbar menu pops up when **both** mouse buttons are pressed at once (or the **middle** button is pressed on a three-button mouse) while the mouse cursor is on the scrollbar. This menu allows several modes particular to the scrollbar to be set.

## Scrollbar

The **xterm** command supports an optional scrollbar composed of a scroll button displayed at the top of the scrollbar and a scroll region at the bottom. The scrollbar is hidden until its display is requested. Pressing both buttons on the mouse at once (or the middle button on a three-button mouse) while the cursor is in any part of the scrollbar displays the scrollbar menu.

The *scroll region* displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved, the size of the highlighted area decreases. Clicking either the left or right button while the mouse cursor is in the scroll region positions the top of the display window at the mouse cursor.

The scroll button causes the window to scroll up and down within the saved text. Clicking the left button moves the window position up (the text scrolls downward), while clicking the right button moves the window position down (the text scrolls upward). The amount of scrolling is modified by the **Shift** and **Ctrl** keys. If neither key is pressed, the window scrolls a single line at a time. Pressing the **Shift** key causes the text to scroll a full window at a time, minus one line. Pressing the **Ctrl** key causes the text to be positioned at the extreme top or bottom of the file.

## HFT Emulation Summary

The **xterm** command supports a window that is equivalent to an HFT virtual terminal.

The following is a summary of HFT emulation functions:

- A subset of HFT ioctl/VTDs is supported. For more information, see "xterm HFT Functions" on page 5-4.

- Keyboard mapping is defined by **XLookupMapping** and the **keycomp** command. For more information, see "**keycomp**" on page 3-12 and "XLookUpMapping" on page 4-69. For information on keyboard mapping, see *Keyboard Description and Character Reference*.

- International Character Support is provided for code page switching via single-shift control characters. For more information on the datastream, see *AIX Operating System Technical Reference*.

- Mouse reports are supported.

- The HFT datastream as defined in *AIX Operating System Technical Reference* is supported.

- Mouse reports are supported.

- HFT escape sequences beyond the standard VT102 set are implemented. For more information, see "xterm Datastream Support" on page 5-8.

### VT102 Emulation Summary

The **xterm** command emulates a VT102 terminal when the **−v** command option is specified. When VT102 emulation is requested, **xterm** sets the **TERM** environment variable to **vt100**.

Five keyboard states are handled by **XLookupMapping**. In order to have VT102 keyboard mapping, a VT102 **.Xkeymap** file must reside in the HOME directory of the user or in a directory supported by **XLookupMapping**.

The VT102 emulation does not support a blinking character attribute nor double-wide and double-size character sets. Also, International Character Support is not provided during VT102 emulation.

# Flags

If an option begins with a + (plus sign) instead of a − (minus sign), the option is restored to its default value. These options override those set in the **.Xdefaults** file.

The **xterm** command uses the following flags:

| | |
|---|---|
| **−ar** | Turns on the auto-raise mode of **xterm**, which automatically raises the specified window when the mouse cursor enters the window. |
| | The default for this option is off. |
| **−b** *num* | Sets the width in pixels of an inner border. The inner border is the distance between characters and the border of the window. The default value for this option is **1**. |
| **−bd** *color* | Determines the color of the highlighted border on color displays. The default value for this option is black. |
| **−bg** *color* | Determines the color of the background on color displays. The default value for this option is white. |
| **−bi** | Defines the icon as a miniature window rather than the default static bitmap. The default value for this option is off. |
| **−bw** *num* | Specifies the width in pixels of the window border. The default value for this is 2. |
| **−cr** *color* | Determines the color of the text cursor on color displays. The default value for this option is black. |

# xterm

| | |
|---|---|
| **−cu** | Causes certain curses applications to display leading tabs correctly. The default value for this option is off. |
| **−dw** | Causes the mouse cursor to move automatically near the center of the window when the **xterm** icon is shown. The default value for this option is off. |
| **−e** *cmd* | Specifies a command to be executed in the window. This can be used in place of starting a shell. If this flag is used, the command and its arguments (if any) must appear last on the **xterm** command line. |
| **−fb** *font* | Specifies a font for use instead of the default bold font. This font must be the same height and width as the normal font. |
| **−fg** *color* | Determines the color of the text on color displays. The default value for this option is black. |
| **−fi** *font* | Specifies the default font to be used for the miniature icon windows. The default value for this option is **Rom6.500** for HFT mode. In VT102 emulation, the default is **nil2**. |
| **−fn** *font* | Specifies a font for use instead of the default font. Any fixed-width font can be used. In HFT emulation, the default is either **Rom14.500** or **Rom10.500**, depending on the size of the display. In VT102 emulation, the default is **vtsingle**. |
| **−ft** *font* | Specifies a font for use in the title bar instead of the default font. The default is the normal display font. |
| **=** *geometry* | Specifies the location and the dimensions of a window. The default for this option is **=80x25+0+0**. |
| | For more information on *geometry*, see "Geometry Specification" on page 3-4. |
| **#** *geometry* | Specifies the location of an icon. If specified, *width* and *height* are ignored. |
| | For more information on geometry specifications, see "Geometry Specification" on page 3-4. |
| **−i** | Causes **xterm** to display the icon rather than the normal window when the window is opened. The default value for this option is off. |
| **−ib** *file* | Causes **xterm** to read a bitmap file and to use the resulting bitmap as the icon. |
| **−j** | Causes **xterm** to move multiple lines up at once if many lines are queued for display. The VT100 escape sequences for smooth scroll can be used to enable or disable this feature from a program. Also, |

the `Mode` menu can be used to set this option interactively.  The default value for this option is off.

**−l**

Appends input from the window to the end of the **logfile** file.  The default is off.  This does not override **LogInhibit** in the **.Xdefaults** file.  For more information about **LogInhibit**, see 2-9.

**−lf** *file*

Specifies the file in which the log is written.  This file is used instead of the default file **XtermLog.**_xxxxx_, where _xxxxx_ is the process ID of **xterm**.  The file is created in the directory where **xterm** is started, or in the home directory for a login **xterm**.  If the file name begins with a | (pipe symbol), the rest of the string is interpreted as a command to be executed by the shell and a pipe is opened to the process.

**−ls**

Causes the shell run under **xterm** to be a login shell.  The user's **.profile** file is read and the initial directory is the home directory.  The default is off.

**−mb**

Turns on the right margin bell.  The default for this option is off.

**−ms** *color*

Determines the color of the mouse cursor on color displays.  The default value of this option is the color of the text cursor.

**−n** *name*

Specifies a window name for use by a window manager.  This name is displayed in the title bar.

*name:number*

Identifies the hostname and display number where **xterm** is to run.  Normally, **xterm** gets the hostname and display number from the environment variable **DISPLAY**.  For more information, see "Display Specification" on page 3-7.

**−nb** *num*

Specifies the right margin distance at which the margin bell rings.  The default value for this option is **10**.

**−po** *num*

Specifies a new page overlap.  Normally, in page scroll mode, a ***page*** is defined as the number of lines in the scrolling region minus the page overlap.

The default value for page overlap is **1**.

**−ps**

Turns the page scroll mode on.

After a page of lines is displayed, **xterm** stops displaying new lines and the text cursor disappears.  Entering **Return** displays one new line.  Pressing the space bar or a character key displays a new page.

**−r**

Reverses the foreground and background colors.  This becomes the normal video mode.

| | |
|---|---|
| **−rw** | Turns on reverse-wraparound mode. The default is off. |
| | This mode allows the cursor to wraparound from the leftmost column to the rightmost column of the previous line. This can be useful in the shell to allow erasing characters backwards across the previous line. |
| **−s** | Turns off synchronous scrolling on the display. The default for this flag is on. |
| | When this flag is specified, **xterm** no longer attempts to keep the screen current while scrolling and can run faster when network latencies are very high. |
| **−sb** | Causes the scrollbar to be displayed during startup and turns on the saving of lines scrolled off of the window. The default for this option is off. |
| **−si** | Normally, the window is repositioned automatically at the bottom of the scroll region after input is processed when you use the scrollbar to review previous lines of text. **−si** disables window repositioning on input when set to off. The default for this option is on. |
| **−sk** | Causes the window to be repositioned automatically in the normal position at the bottom of the scroll region when a key is pressed. This option is intended for use with the scrollbar to review previous lines of text. |
| | The default for this option is off. |
| **−sl** *num* | Specifies the maximum number of lines to save that are scrolled off of the top of the window. The default value for this option is **64**. |
| **−sn** | Causes the status line to be displayed in normal video (the status line is still enclosed in a box). When displayed, the status line by default appears in reverse-video relative to the rest of the window. |
| **−st** | Causes the status line to be displayed on startup. The default is no display of the status line on startup. |
| **−tb** | Disables the display of the title bar on startup. By default, the title bar is displayed on startup. |
| **−ti** | Causes the display of the window name to the right of the static bitmap icon. The default is off. |
| | Normally in the icon, the window name is under the bitmap. |
| **−v** | Enables VT102 emulation. If this option is not given, an HFT terminal is emulated. |
| | **Note:** Keyboard map is needed for this mode. |

| | |
|---|---|
| $-$**vb** | Turns on the visual bell mode that flashes the **xterm** window on receipt of the **Ctrl-G** key combination. The default of this option is off. |
| $-$**W** | Causes the mouse cursor to be placed in the middle of the **xterm** window when the window is created. The default is not to move the mouse cursor. |
| $-$**132** | Causes the **smrm** escape sequences to be recognized and the **xterm** window to be resized as specified. Normally, the **sm/rm** escape sequences that switch between the 80 and 132 column modes are ignored. The default for this flag is off. |

## Default Keywords

The default keywords for use with the **xterm** command are:

> **ActiveIcon**
> **AutoRaise**
> **Background**
> **BodyFont**
> **BoldFont**
> **Border**
> **BorderWidth**
> **C132**
> **Cursor**
> **DeIconifyWarp**
> **Foreground**
> **Geometry**
> **IconBitmap**
> **IconFont**
> **IconStartup**
> **InternalBorder**
> **JumpScroll**
> **LogFile**
> **Logging**
> **LogInhibit**
> **MarginBell**
> **Mouse**
> **NMarginBell**
> **PageOverlap**
> **PageScroll**
> **ReverseVideo**
> **ReverseWrap**
> **SaveLines**
> **ScrollBar**
> **ScrollInput**
> **ScrollKey**

StatusLine
StatusNormal
TextUnderIcon
TitleBar
TitleFont
VisualBell
Warp

For more information about the use of these keywords, see "Changing X-Windows Defaults" on page 2-4.

# Chapter 4.  Programming Interface to X-Windows

# CONTENTS

# About This Chapter

This chapter describes the C programming language interface to X-Windows.

## What You Need to Know

In order to use this chapter, you should be familiar with, and be able to write programs in, the C programming language. For more information, see *C Language Guide and Reference*.

## How This Chapter Is Organized

- General Reference Information — This part of the chapter provides an overview and describes some of the characteristics of the X-Windows functions. It also provides general reference information that is common to a group of program interface functions.

- Subroutine Reference Information — This part of the chapter describes each of the X-Windows program interface functions. The descriptions are in alphabetical order by subroutine name.

If you are a first-time user, you should read and become familiar with the general reference information before attempting to use the subroutine reference information.

# General Reference Information

The programming interface to X-Windows is a set of **C** language subroutines that you can use to create, destroy, and manipulate windows and the data and graphics associated with the windows.

## Subroutines

The following is a list of the X-Windows subroutines grouped according to the type of function provided.

- Opening and Closing Displays

    **XOpenDisplay** – Open a display.
    **XCloseDisplay** – Close a display
    **XSetDisplay** – Set the current display connection.

- Creating and Destroying Windows

    **XCreate** – Create and place a window.
    **XCreateTerm** – Create and place text related window.
    **XCreateTransparency** – Create an unmapped, transparent window.
    **XCreateTransparencies** – Create multiple unmapped transparent windows.
    **XCreateWindow** – Create an unmapped, opaque window.
    **XCreateWindows** – Create multiple unmapped, opaque windows.
    **XCreateWindowBatch** – Create multiple opaque or transparent windows.
    **XDestroySubwindows** – Destroy all subwindows of specified window.
    **DestroyWindow** – Unmap and destroy window and all subwindows.

- Manipulating Windows

    **XChangeWindow** – Change window size.
    **XCircWindowDown** – Lower the highest mapped child of the specified window.
    **XCircWindowUp** – Raise the lowest mapped child of the specified window.
    **XConfigureWindow** – Change window size and location.
    **XLowerWindow** – Lower a window.
    **XMapSubwindow** – Map all subwindows of the specified window.
    **XMapWindow** – Map a window.
    **XMoveWindow** – Move and raise a window.
    **XRaiseWindow** – Raise a window.
    **XUnmapSubwindows** – Unmap all subwindows of specified window.
    **XUnmapTransparent** – Unmap a transparent window.
    **XUnmapWindow** – Unmap a window.

- Receiving Status and Changing Modes

    **XFetchName** − Set a pointer to a window name.
    **XChangeBackground** − Change the background of a window.
    **XChangeBorder** − Change and repaint a window border.
    **XClearIconWindow** − Clear an icon window.
    **XClipClipped** − Set clip-mode to clipped.
    **XClipDrawThrough** − Set clip-mode to draw-through.
    **XCondWarpMouse** − Move mouse relative to the destination window.
    **XGetResizeHint** − Assign window size parameters to client variables.
    **XInterpretLocator** − Convert absolute window coordinates to relative.
    **XQueryMouse** − Determine the current mouse coordinates.
    **XQueryMouseButtons** − Gives current mouse coordinates and button states.
    **XQueryTree** − Determine a window tree structure.
    **XQueryWindow** − Determine information about a window.
    **XSetIconWindow** − Set an icon Window.
    **XSetResizeHint** − Give X-Windows a window shape hint.
    **XStoreName** − Assign a name to a window.
    **XTileAbsolute** − The background is tiled relative to the window origin.
    **XTileRelative** − The background is tiled relative to the closest parent window.
    **XUpdateMouse** − Functions like **XQueryMouse** but also reads events.
    **XWarpMouse** − Move the mouse relative to the destination window.

- Drawing Lines and Filling Areas

    **XAppendVertex** − Append Vertices to the output buffer.
    **XClearVertexFlag** − Clear the state of the append-vertex flag marking.
    **XDraw** − Draw an arbitrary polygon or curve.
    **XDrawDashed** − Functions as **XDraw** except the line is dashed.
    **XDrawFilled** − Functions as **XDraw**, filling the shape with a pixel value.
    **XDrawPatterned** − Functions as **XDraw** using alternate pixel values for the line.
    **XDrawTiled** − Functions as **XDraw** and fills the shape with a tile pixmap.
    **XLine** − Draw a line between two window coordinates.
    **XMakePattern** − Make a pattern according to specified bit string.
    **XQueryBrushShape** − Return closest supported shape to specified shape.

- Controlling the Raster

    **XClear** − Clear the window and repaint the background.
    **XCopyArea** − Copy a region to another region of the same window.
    **XMoveArea** − Functions as **XCopyArea** with *planes = allplanes*.
    **XPixFill** − Do a display function in a region of a window.
    **XPixmapPut** − Do a display function in a an area of a pixmap and window.
    **XPixSet** − Window area (all planes) set to specified pixel value, no clipping mask.
    **XQueryTileShape** − Return the closest supported tile shape.
    **XTileFill** − Perform a display function in a window region using a tile pixmap.
    **XTileSet** − Functions as **XTileFill** with all planes and no clipping mask.

- Moving Bits and Pixels

  **XBitmapBitsPut** − Copy a client-supplied bitmap into a window region.
  **XPixmapBitsPutXY** − Copy a client-supplied pixmap into a window.
  **XPixmapBitsPutZ** − Copy a client-supplied pixmap into a window.
  **XPixmapGetXY** − Return a pixmap into the specified area of memory.
  **XPixmapGetZ** − Return a pixmap into the specified area of memory.
  **XPixmapSave** − Create a pixmap from the specified portion of a window.

- Storing and Freeing Maps

  **XCharBitmap** − Create a bitmap from the specified font character.
  **XFreeBitmap** − Free all storage associated with the specified bitmap.
  **XFreePixmap** − Free all storage associated with the specified pixmap.
  **XMakePixmap** − Create a pixmap from a bitmap and two pixel values.
  **XMakeTile** − Create a pixmap to use for tiling.
  **XStoreBitmap** − Create a bitmap for later use.
  **XStorePixmapXY** − Create a pixmap for later use.
  **XStorePixmapZ** − Create a pixmap for later use.

- Using the Cursor

  **XCreateCursor** − Create a cursor.
  **XDefineCursor** − Use the specified cursor when the mouse is in the window.
  **XFreeCursor** − Free all storage associated with the specified cursor.
  **XQueryCursorShape** − Return the closest supported cursor shape.
  **XStoreCursor** − Create and store a cursor.
  **XUndefineCursor** − Use the parent window's cursor.

- Using Color Maps

  **XFreeColors** − Free color map cells.
  **XGetColor** − Return a color structure for the specified color name.
  **XGetColorCells** − Allocate color map cells.
  **XGetHardwareColor** − Return the hardware color closest to the one specified.
  **XParseColor** − Convert a color string to format suitable for future color calls.
  **XQueryColor** − Return the color values for the specified pixel value.
  **XQueryColors** − Return color values for each specified definition.
  **XStoreColor** − Return the closest available color for the specified pixel value.
  **XStoreColors** − Return the closest available colors for the specified pixel values.

- Using Fonts

  **XCharWidths** − Determine the width, in the specified font, of each character in a string.
  **XCloseFont** − Deallocate all storage associated with the specified font.
  **XFontWidths** − Return a pointer to an array with the width of every font character.
  **XFreeFont** − Notify the X Server the specified font is no longer needed.
  **XGetFont** − Load the font with the specified name.

**XOpenFont** – Performs the functions **XGetFont, XQueryFont,** and **XFontWidths** in one operation.
**XQueryFont** – Retrieve various facts about a font.
**XQueryWidth** – Width in pixels of a null-terminated string.
**XStringWidth** – Width of a null-terminated string with supplied font information.

- Writing and Reading Text

  **XText** – Draw text into a window using the specified font and display function.
  **XTextPad** – Functions as **XText** with character spacing specified.
  **XTextMask** – Functions as **XText**, but uses font as a mask.
  **XTextMaskPad** – Functions as **XTextMask** with character spacing specified.

- Controlling Access

  **XAddHost** – Add a host to those allowed to open connections to this display.
  **XGetHosts** – Return the current list of hosts allowed to open connections.
  **XRemoveHost** – Remove the host from those allowed to open connections.

- Handling User Preferences

  **XAutoRepeatOn** – Turn the keyboard auto-repeat on.
  **XAutoRepeatOff** – Turn the keyboard auto-repeat off.
  **XFeep** – Ring a bell on the terminal.
  **XFeepControl** – Define the base volume for **XFeep**.
  **XGeometry** – Determine a window placement.
  **XGetDefault** – Return default window options.
  **XKeyClickControl** – Set the volume of the keyboard click.
  **XLockToggle** – Do not send **keypressed** or **keyreleased** for the shiftlock key.
  **XLockUpDown** – Send **keypressed** and **keyreleased** for the shiftlock key.
  **XMouseControl** – Define how the mouse moves.
  **XParseGeometry** – Parse a window geometry string.
  **XReadBitmapFile** – Read a bitmap file.
  **XScreenSaver** – How many idle minutes before the screen is blanked.

- Copying and Pasting

  **XAppendBuffer** – Append bytes to the specified buffer.
  **XFetchBuffer** – Fetch the contents of the specified buffer.
  **XFetchBytes** – Fetch the contents of buffer 0.
  **XRotateBuffers** – Rotate the buffers
  **XStoreBuffer** – Store a string of bytes in the specified buffer.
  **XStoreBytes** – Store a string of bytes in buffer 0.

- Handling Events

  **XCheckMaskEvent** – Check for events specified in mask.
  **XCheckWindowEvent** – Check for specified events from specified windows.
  **XCompressEvents** – Suppress all but the last mousemoved events.
  **XExpandEvents** – Supply all mousemoved events.
  **XFlush** – Send all output requests that have been buffered but not yet sent.

**XMaskEvent** – Look for event specified in mask.
**XNextEvent** – Get the next event from the queue.
**XPeekEvent** – Look at the next event on the queue without removing it.
**XPending** – Return the number of input events still in the queue.
**XPutBackEvent** – Put an event back on the head of the input event queue.
**XSelectInput** – Define which input events the window is interested in.
**XSync** – Flush the buffer and wait until all events and errors are processed.
**XWindowEvent** – Look for specified events from a specified window.

- Associating Resources to Structures

    **XCreateAssocTable** – Create an association table.
    **XDeleteAssoc** – Delete an association in the specified table.
    **XDestroyAssocTable** – Deallocate the specified association table.
    **XLookUpAssoc** – Retrieve data stored in the specified association table.
    **XMakeAssocTable** – Insert data into the specified association table.

- Controlling the Mouse, Buttons, and X Server

    **XGrabButton** – Grab the mouse when the button specified in this call is pressed.
    **XGrabMouse** – Mouse events go to windows with **XSelectInput** calls.
    **XGrabServer** – Retain exclusive control of the X Server for this connection only.
    **XUngrabButton** – Notify the X Server the client no longer needs the mouse.
    **XUngrabMouse** – Release the mouse if it was grabbed by **XGrabMouse**.
    **XUngrabServer** – Free the X Server for other connections.

- Controlling the Keyboard

    **XFocusKeyboard** – Designate the specified window as the input focus window.
    **XLookupMapping** – Return the length of keyboard events.
    **XRebindCode** – Change the keyboard binding.
    **XUseKeymap** – Change keymap files.

- Handling Errors

    **XErrorHandler** – Handle **XError** events.
    **XIOErrorHandler** – Handle catastrophic errors.
    **XErrDescrip** – Return a null-terminated string describing the error code.

# Compiling X Programs

Before attempting to program to the X-Windows interface, note that you must install the **Sockets** program from the Multi-User Services diskettes of the AIX Operating System. For information on installing programs from Multi-User Services, see *Installing and Customizing the AIX Operating System*.

The following compiler command can be used to build your program:

**cc** {*compiler options*} **-o**samples sample**.c -lX -lsock**

In the preceding example, *sample*.**c** is the name of your C language source program, *sample* is the name of your executable C program, **lX** is the X-Windows subroutine library (**/usr/lib/libX.a**) and **lsock** is the AIX sockets library (**/usr/lib/libsock.a**).

The compiler *definitions* for structures, parameters, error codes, and data types are located in **/usr/include/X/Xlib.h** and **/usr/include/X/X.h**. You must include this file in any program that uses X-Windows subroutines. To do so, insert the following statements early in your program:

```
#include <X/Xlib.h>   /* also includes X/X.h */
```

## System Structure

The application program you create is the *client* part of a client-server relationship; you write the program and the X Server provides it independence from the hardware. On an RT PC, one to four physical displays can function as a maximum of 16 virtual terminals. There is one X Server for each virtual terminal that runs X-Windows. In this chapter the term *display* means a logical virtual terminal with its associated keyboard, mouse, and server unless it is explicitly stated otherwise. The following diagram shows the client-X Server-display relationship:
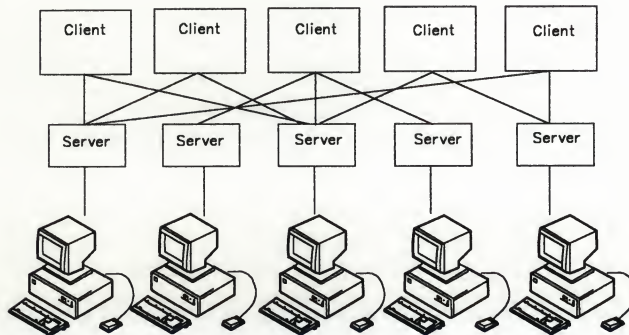


Note that each client can interact with many X Servers and each X Server can interact with many clients.

The programming interface is based on a network protocol that allows your program (client) to efficiently interact with displays connected to other processors in a network.

# Coordinates

Like the display screen, each window has its own XY coordinate system with the origin at the upper left hand corner. The X axis is horizontal from left to right; the Y axis is vertical from top to bottom. Each addressable point on the screen is called a *pixel* and corresponds to one XY point in the coordinate system. Because each window has its own coordinate system, operations within windows can be insensitive to the window position on the screen.

The origin of a window is inside the border, if it has one, and window size is always the usable number of pixels within the border. The window border is maintained by the X Server and output to the window is clipped so as not to extend into the border.

# Windows

All windows on a display are in a strict parent-child hierarchy beginning with the root window. The root window covers the entire display and is the only window without a parent window. Windows with the same parent window are considered *sibling* windows.

There are two type of windows, *opaque* and *transparent*. Opaque windows have borders, and they also have a background pattern, or color, called a *tile*. You specify the width and color for the border, and the pattern or color for the tile.

An opaque window's background is tiled with a pattern, which is usually a constant color. The pattern can be either relative to the parent (the pattern is shifted appropriately to match the parent window) or absolute (the pattern is positioned in the window independently of the parent window. This positioning is called the *tile mode* of the window and can be set with the appropriate function calls.

Sending output to a transparent window is equivalent to sending it to the parent window, except that the transparent window's clip mode is used instead of the clip mode of the parent window and the output is clipped to the boundaries of the transparent window.

If an opaque window is stacked on top of another window, the covered area is obscured for both input and output. Attempts to display output to the covered area do nothing, and input events from the mouse are not generated.

Transparent windows do not have borders, and they obscure other windows only from input. If you move a transparent window, it has no effect on the display.

For a given window its subwindows can be stacked in any order, like papers on a desktop, with arbitrary overlaps. If window w1 partially or completely covers window w2, then w1 obscures w2. Window hierarchies never interleave; if window w1 obscures sibling window w2, then subwindows of w2 never obscure w1 or subwindows of w1.

A window is not restricted in size or placement by the boundaries of its parent, but a window is always visibly clipped by its parent: portions of the window that extend outside the boundaries of the parent are never displayed and do not obscure other windows.

A window can be either *mapped* or *unmapped*, and an unmapped window is never visible on the screen. A mapped window can only be visible if all of its ancestors are also mapped.

Output to a window with no subwindows is always clipped to the visible portions of the window, and output to such a window never extends into obscuring windows. Output to a window that contains subwindows can be performed in either of two modes. In *clipped* mode the output is clipped normally by all obscuring windows including subwindows. In *draw-through* mode the output is not clipped by subwindows. For example, draw-through mode is used on the root window during window management when tracking the mouse with the outline of a window to indicate how the window is to moved or resized. If clipped mode was used instead, the entire outline would not be visible.

## Creating Windows

**XCreate** and **XCreateTerm** process geometry specifications. If a sufficiently complete geometry specification (see **XParseGeometry** and **XGeometry**) is passed in, the window is created automatically. If no X or Y locations are set by the geometry spec, the user is prompted to interactively position the window. A prompt window is popped up in the upper left hand corner, and the mouse is grabbed. For **XCreateTerm**, the prompt window is appended by the height and width of the **fwidth** and **fheight** units.

The following **MakeWindow** X defaults control the appearance of a prompt window and the cursor to be used when rubber banding:

- BodyFont
- ReverseVideo
- BorderWidth
- InternalBorder
- Freeze
- Foreground
- Background
- Border
- Mouse
- MouseMask.

If **Freeze** is set, the server is *frozen* while the window is being created. So for example, one of these might be specified as **.MakeWindow.Freeze** in your **.Xdefaults** file. A box the size of the minimum window is rubber-banded on the screen.

If the left button is pressed, the outline of the default window at the mouse's current position of the default size is shown; when the button is released, the window is created.

If the right button is pressed, the outline of the default window at its default size and the current location of the mouse is shown; when the button is released, the window's upper-left corner is created at the current cursor location. If using **XCreateTerm** and **MakeWindow.ClipToScreen** is set, the window is clipped to the screen unless the minimum size requirements preclude it.

If the center button is pressed, it indicates one corner of the window should be set at the current mouse location. When the center button is released, the window is created with the other corner of the window at the current mouse location, unless the minimum size has not been met.

The following structures allow efficient creation of many similar windows or subwindows simultaneously (a function often wanted when creating menus or complex forms):

```
typedef struct _OpaqueFrame {
    Window self;                       /* window ID of the window, filled in later */
    short x, y;                        /*where to create the window */
    short width, height;              /* window size */
    short bdrwidth;                   /* border width */
    Pixmap border, background;        /* border pixmap */
} OpaqueFrame;

typedef struct _TransparentFrame {
    Window self;                       /* window ID of the window, filled in later */
    short x, y;                        /* where to create the window */
    short width, height;              /* window size */
} TransparentFrame;

typedef struct _BatchFrame {
    short type;                        /* one of (IsOpaque, IsTransparent) */
    Window parent;                     /* window ID of the window's parent */
    Window self;                       /* window ID of the window, filled in later */
    short x, y;                        /* where to create the window */
    short width, height;              /* window width and height */
    short bdrwidth;                   /* window border width */
    Pixmap border;                     /* window border pixmap */
    Pixmap background;                 /* window background pixmap */
} BatchFrame;
```

## Pixels and Planes

Each pixel on a monochrome (black and white) display has one bit of information associated with it; the bit is either 0 (for black) or 1 (for white).

Color displays, however, need multiple bits per pixel in order to properly regulate the red, green, and blue color guns that provide the full range of visible colors available to the particular display. For example, assume a display has 4 bits per pixel, numbered 0, 1, 2, and 3. The display is said to be 4 planes deep. One plane contains all the bit 0s, the second the bit 1s, the third the bit 2s, and the fourth the bit 3s.

## Bitmaps

A *bitmap* is a rectangle of bits that has a width in pixels and a height in pixels. A bitmap has a depth of 1 plane. The following shows a bitmap 17 pixels wide by 8 pixels high.
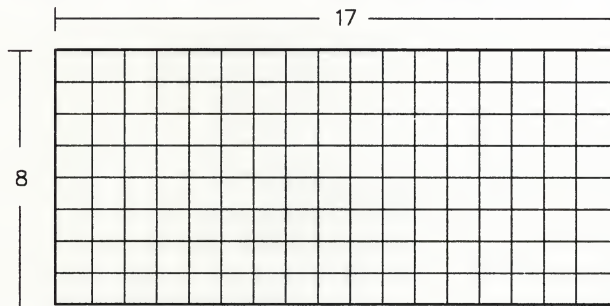
A bitmap is represented in storage according to the following:

- The map is stored in rows, top row to bottom, each row beginning on a 16-bit word boundary.

- Each row is padded, if necessary, to a multiple of 16 bits.

- The size of the bitmap (bytes) is:

    $$((\text{width} + 15) \div 16)*(\text{height})*(2) = \text{size in bytes}$$

- You can use the **BitmapSize** macro to compute the size.

The bit order in each 16-bit word is exactly opposite to the bit order on the display. The least-significant bit of the word (bit 0), is the leftmost visible pixel on the display. Note that there is a supplied bit-reversal function (**XRevShorts**) described in "IBM-Specific X-Windows Implementation" on page 4-108.

# Pixmaps

A *pixmap* is a rectangle of pixels that has a width in pixels and a height in pixels. A pixmap is as deep as the pixel has bits. Pixmaps can be represented in storage in either *XY format* or *Z format*.

In XY format, each plane in the pixmap is represented as a bitmap; the bitmaps appear in storage from most significant to least significant in sequence. The following shows a pixmap in XY format:
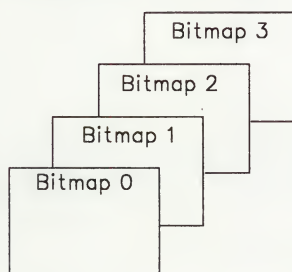


The size of the pixmap in bytes is:

$$((\text{width} + 15) \div 16)*(\text{height})*(\text{depth})*(2) = \text{size in bytes}$$

The **XYPixmapSize** macro computes the size of an XY format pixmap.

In Z format, the pixels are stored row by row, top to bottom, left to right within the row. The following shows a pixmap in Z format:

For a display with 2-8 planes, each pixel value is represented as a single byte. The pixmap size in bytes is:

(width)*(height) = size in bytes    /* Z format, 2-8 planes */

Use the **BZPixmapSize** macro to compute the size.

For a display with 9-16 planes, each pixel value is represented by a 16-bit word. The pixmap size in bytes is:

(width)*(height)*(2) = size in bytes   /* Z format, 9-16 planes */

Use the **WZPixmapSize** macro to compute the size.

# Display Operations

You must connect your program (client) to the X Server before performing any operations. To connect, use **XOpenDisplay** to open the display. You provide the display name (as specified in "Display Specification" on page 3-7) and X-Windows returns a pointer to a Display structure (defined in **Xlib.h**) that contains information about the display such as:

- The network socket
- The root window ID
- The number of display bit planes
- The number of display color map cells
- Event queue information
- Output buffer information
- The display name you provided in the open function
- The width and height of the display.

X-Windows assumes you are working with a single display at a time, and that your program keeps track of the current display. You can change the current display with the **XSetDisplay** function.

To disconnect from the X Server when your program is finished with the display, use the **XCloseDisplay** subroutine and close the display. Resource ids (window, font, bitmap, pixmap or cursor) created with this display are destroyed and should not be used after the display is closed. Output events that have been buffered but not yet sent are discarded. The effect of **XCloseDisplay** is automatically achieved if a process exits.

If you call **fork** after opening a display, you should take the following precautions when dealing with the connection to the window system:

- Flush the output buffer before waiting on the child process.
- Make sure you have processed all input events before calling **fork** from, or exiting from, the child process.

If you do not follow the above procedures, your program could perform the same operations twice.

# Window Operations

A newly created window is not automatically displayed; to display a window, call **XMapWindow**. New windows do not have cursors defined; the cursor is that of the parent window until a cursor is registered. A window is not visible on the screen until it and all of its ancestors are mapped and it is not obscured by any of its ancestors. Any output to a window that is not visible is discarded. When an opaque window is mapped, an exposure event is generated. Mapping an already mapped window has no effect and does not raise the window.

If the window is opaque, **XMapWindow** generates **ExposeWindow** events (see "Events") on each opaque window that it causes to become displayed. If your program first maps the window, then paints the window, then begins processing input events, the window ends up painted twice. To avoid this your program should either:

- First map the window, then call **XSelectInput** for exposure events, then repaint the window explicitly.

- First call **XSelectInput** for exposure events, then process input events normally.

The event list includes **ExposeWindow** events for each window that has appeared on the screen and the normal response from your program to **ExposeWindow** should be to repaint the window. The second method is preferred because it usually leads to simpler programs.

X-Windows does not take responsibility for the contents of windows; your program must be responsible. When part (or all) of a window is hidden and then brought back onto the screen, an exposure event is generated to notify your program to restore the contents of the window. Your program should be prepared to regenerate windows on demand.

# Events

Input from X-Windows is in the form of events. Events can be the result of a command or they can be completely asynchronous (such as a keyboard event). Your program must specify which events it wants to receive because events that your program does not ask for are not sent to your program.

Definitions for the event IDs are in **X.h**; definitions for the associated event structures are in **Xlib.h**. **Xlib.h** also contains a generic structure called **XEvent** that contains information common to all events. When an event occurs you can use this structure until you determine the correct event type. The **XEvent** structure is defined as follows:

```
typedef struct _XEvent {
    unsigned long type;          /* type of event */
    Window window;               /* window selceting this event */
    long pad_11;                 /* event-specific data */
    long pad_12;                 /* event-specific data */
```

```
        Window subwindow;         /* child window (if any) event occurred in */
        long pad_14;              /* event-specific data */
} XEvent;
```

When an event occurs, X-Windows sends it to the smallest enclosing window for which a client has selected input for that event type. If no window in the hierarchy has selected input for that event type, it is eventually discarded.

If your program is receiving both **ButtonPressed** and **ButtonReleased** events on a window and a **ButtonPressed** event occurs, the mouse is automatically grabbed until all buttons are released.

If the mouse has been grabbed, then the following events go only to windows for which the grabbing client has called **XSelectInput**:

- **ButtonPressed**
- **ButtonReleased**
- **MouseMoved**
- **EnterWindow**
- **LeaveWindow**

If the client has not called **XSelectInput** on the window where the event would normally be sent, the window where the original **ButtonPressed** event occurred receives the event but only if the event type is being received for that window and the type is neither **XEnterWindow** nor **XLeaveWindow**.

Input events (such as **KeyPressed** and **MouseMoved**) arrive asynchronously from the server and are queued until requested by a call to **XNextEvent** or **XWindowEvent**. Some of the functions, including **XChangeWindow** and **XRaiseWindow**, generate *exposure* events. An exposure event is a request to repaint sections of a window that do not have valid contents. These events also arrive asynchronously, but the client can explicitly wait for them by calling **XSync** after calling a function that may generate exposure events.

## Key/Button Events

The structure used with key or button events is defined as follows:

```
struct _XKeyOrButtonEvent {
    unsigned long type;       /* KeyPressed, ButtonReleased, and so on */
    Window window;            /* which window selected this event      */
    unsigned short time;      /* in 10 ms. ticks for button events     */
    short detail;             /* event-dependent data                  */
    short x;                  /* mouse x coordinate within event window */
    short y;                  /* mouse y coordinate within event window */
    Window subwindow;         /* child window (if any) mouse was in     */
    Locator location;         /* absolute coordinates of mouse          */ } ;
```

For events that contain mouse coordinates, the coordinates are relative to the event window, even if the mouse is not in the window (because of mouse grabbing or keyboard

focusing). If the mouse is also in a descendant of the event window, the subwindow is set to that descendant; otherwise the subwindow is 0. The locator defines the mouse coordinates in absolute terms and you can use it as an argument to **XInterpretLocator**.

A time value is present only for the **ButtonPressed** and **ButtonReleased** events.

The time value consists of 16 bits and it wraps after approximately 11 minutes.

For event structures that contain the detail definition (two bytes), the high-order byte contains the state of various keys and buttons just before the event (see **XLib.h**) and is defined as follows:

| | | |
|---|---|---|
| **AltGraphMask** | 0x8000 | Alt-graphics key |
| **ControlMask** | 0x4000 | Control key |
| **MetaMask** | 0x2000 | Meta (symbol) key |
| **ShiftMask** | 0x1000 | Shift key |
| **ShiftLockMask** | 0x0800 | ShiftLock key |
| **LeftMask** | 0x0400 | Left button |
| **MiddleMask** | 0x0200 | Middle button |
| **RightMask** | 0x0100 | Right button |

For **KeyPressed** and **KeyReleased** events, the low-order byte of the detail gives the keycode (not the ASCII code). The file **X/Xkeyboard.h** contains macros useful to classifying keycodes. The tests **IsShiftKey( )**, **IsCursorKey( )**, **IsKeypadKey( )**, **IsFunctionKey( )**, **IsPFKey( )**, and **IsTypeWriterKey( )** can be used with a keycode to classify the grouping of a key. Typically, a client uses **XLookupMapping** to determine the string (if any) associated with a key.

For **ButtonPressed** and **ButtonReleased** events, the low-order detail byte is one of the following:

| | |
|---|---|
| **RightButton** | 0 |
| **MiddleButton** | 1 (both buttons) |
| **LeftButton** | 2 |

Key/button events are defined as follows:

| Event Type | Hex Code | Result |
|---|---|---|
| **KeyPressed** | 0x0001 | Keyboard key pressed |
| **KeyReleased** | 0x0002 | Keyboard key released |
| **ButtonPressed** | 0x0004 | Mouse button pressed |
| **ButtonReleased** | 0x0008 | Mouse button released |

## Motion Events

For **EnterWindow** and **LeaveWindow** events, the low-order detail byte is either 0 or one of the following:

**IntoOrFromSubwindow**    1
**VirtualCrossing**           2

**EnterWindow** and **LeaveWindow** events with associated low-order detail byte information are generated as follows:

When the mouse moves from window A to window B, and B is an ancestor of A:

- A receives a **LeaveWindow** event with detail 0.
- Windows between A and B exclusive that have **LeaveWindow** selected receive a **LeaveWindow** event with detail 2.
- B receives an **EnterWindow** event with detail 1.

When the mouse moves from window A to window B, and B is a descendant of A:

- A receives a **LeaveWindow** event with detail 1.
- Windows between A and B exclusive that have **EnterWindow** selected receive an **EnterWindow** event with detail 2.
- B receives an **EnterWindow** event with detail 0.

When the mouse moves from window A to window B, with window C being their least common ancestor:

- A receives a **LeaveWindow** event with detail 0.
- Windows between A and C exclusive that have **LeaveWindow** selected receive a **LeaveWindow** event with detail 2.
- B receives an **EnterWindow** event with detail 0.

At the start of a mouse grab with the button in window A:

- A receives a **LeaveWindow** event with detail 0 if the grabbing client has not issued **XSelectInput** on A.
- Ancestors of A (not including the root window) receive a **LeaveWindow** event with detail 2 if the grabbing client has not issued **XSelectInput** on the window and the window has **LeaveWindow** selected.

At the end of a mouse grab with the mouse in window A:

- Ancestors of A (not including the root window) receive an **EnterWindow** event with detail 2 if the grabbing client has not issued **XSelectInput** on the window, and the window has **EnterWindow** selected.

- A receives an **EnterWindow** event with detail 0 if the grabbing client has not issued **XSelectInput** on window A.

These same **EnterWindow** and **LeaveWindow** event scenarios from the low-order detail byte perspective:

- A detail of 0 means the mouse entered this window from, or left this window towards, some place outside the window's hierarchy.

- A detail of 1 means that the mouse entered this window from, or left this window towards, one of its descendants.

- A detail of 2 means that the mouse moved from a descendant of the window to a place outside the window's hierarchy: or just the opposite.

**EnterWindow** and **LeaveWindow** events with detail 0 or 1 propogate to the smallest enclosing window that has actually selected the event.

The motion events are defined as follows:

| Event Type | Hex Code | Result |
|---|---|---|
| **EnterWindow** | 0x0010 | Mouse entering window |
| **LeaveWindow** | 0x0020 | Mouse leaving window |
| **MouseMoved** | 0x0040 | Mouse moves within window |
| **RightDownMotion** | 0x0400 | Mouse moves with right button down |
| **MiddleDownMotion** | 0x0800 | Mouse moves with middle button down |
| **LeftDownMotion** | 0x1000 | Mouse moves with left button down |

## Exposure Events

The structure used with **ExposeRegion** and **ExposeWindow** events is defined as follows:

```
struct _XExposeEvent  {
    unsigned long type;        /* ExposeRegion or ExposeWindow         */
    Window window;             /* which window selected this event     */
    short pad_s2;              /* in 10 ms. ticks for button events    */
    short detail;              /* 0 or ExposeCopy          */
    short width;               /* width of exposed area        */
    short height;              /* height of exposed area       */
    Window subwindow;          /* child window (if any) actually exposed */
    short y;                   /* top of exposed region            */
    short x;                   /* top left edge of ExposeRegion        */ } ;
```

The structure used with **ExposeCopy** is defined as follows:

```
typedef struct _XExposeCopyEvent  {
    unsigned long type;                    /* ExposeCopyEvent               */
```

```
Window window;                        /* which window selected this event    */
long pad_11;
long pad_12;
Window subwindow;                     /* child window (if any) actually exposed */
long pad_14; } ;
```

When only parts of a window become exposed (such as when an obscuring window is moved), **ExposeRegion** events are sent describing each newly-exposed area. However, if only **ExposeWindow** has been selected, a single **ExposeWindow** event is sent instead. If the region exposure is the result of **XCopyArea** or **XMoveArea** functions, then **ExposeCopy** is set in the detail word. If the exposure is actually that of a child of the window selecting the event, the subwindow is set to that child and the coordinates are actually for the subwindow; otherwise the subwindow is 0. For a given window exposure, **XCopyArea**, or **XMoveArea**, all resulting **ExposeRegion** events are sent contiguously with no other events interspersed.

**ExposeCopy** is only sent to terminate a string of **ExposeRegion** events that might be generated by calls to **XCopyArea** or **XMoveArea**. This is just an acknowledgement that all side effects have been generated; there is no information in the event.

The X Server always clears an exposed area to the background color before sending an exposure event to the your program. Unfortunately, your program may have sent output to the window between the time it is cleared and the time that your program reads the exposure event from the queue. This can lead to difficulty in two cases:

- The exposure event was **ExposeWindow** with new window dimensions. After a size change your program probably wants output to go to a different place.

- Your program is using a display function that is not *invertible*. An invertible function means that applying the function more than once does not have the same effect as applying it just once.

In both cases, your program should explicitly clear the exposed area (paint it with the background) upon receipt of an exposure event *before* doing any repainting. Failure to do so causes undesirable output on the screen.

The exposure events are defined as follows:

| Event Type | Hex Code | Result |
|---|---|---|
| **ExposeWindow** | 0x0080 | Full window changed and/or exposed |
| **ExposeRegion** | 0x0100 | Region of window exposed |
| **ExposeCopy** | 0x0200 | region exposed by **XCopyArea** |

## Miscellaneous Events

The structure used with **UnmapEvent** is defined as follows:

```
typedef struct _XUnmapEvent {
    unsigned long type;           /* UnmapWindow                    */
    Window window;                /* which window selected this event   */
    long pad_11;
    long pad_12;
    Window subwindow;             /* child window (if any) actually unmapped */
    long pad_14; } ;
```

The structure used with **FocusChange** is defined as follows:

```
struct _XExposeCopyEvent {
    unsigned long type;           /* FocusChange                    */
    Window window;                /* which window selected this event   */
    short pad_s1;
    short detail;                 /* EnterWindow or LeaveWindow         */
    long pad_12;
    Window subwindow;             /* child window (if any) of focus change */
    long pad_14; } ;
```

The miscellaneous events are defined as follows:

| Event Type | Hex Code | Result |
|---|---|---|
| **UnmapWindow** | 0x2000 | Window is unmapped |
| **FocusChange** | 0x4000 | Keyboard focus changed |

# Display Functions

When you update a portion of the screen, a display function (*func* parameter) that you specify determines how the destination bits already on the screen are logically combined with the source bits from somewhere else.

There are 16 display functions (defined in **X.h**) as follows:

| Function Name | Hex Code | Logical Operation |
|---|---|---|
| GXclear | 0x0 | 0 |
| GXand | 0x1 | source **AND** destination |
| GXandReverse | 0x2 | source **AND NOT** destination |
| GXcopy | 0x3 | source |
| GXandInverted | 0x4 | (**NOT** source) **AND** destination |
| GXnoop | 0x5 | destination |
| GXxor | 0x6 | source **XOR** destination |
| GXor | 0x7 | source **OR** destination |
| GXnor | 0x8 | (**NOT** source) **AND NOT** destination |
| GXequiv | 0x9 | (**NOT** source) **XOR** destination |
| GXinvert | 0xA | **NOT** destination |
| GXorReverse | 0xB | source **OR NOT** destination |
| GXcopyInverted | 0xC | **NOT** source |
| GXorInverted | 0xD | (**NOT** source) **OR** destination |
| GXnand | 0xE | (**NOT** source) **OR NOT** destination |
| GXset | 0xF | 1 |

If a window has multiple planes, the display function is computed once for each plane.

Seven of the 16 display functions are invertible:

**GXandReverse**
**GXxor**
**GXnor**
**GXequiv**
**GXinvert**
**GXorReverse**
**GXnand**.

# Plane Mask

Many subroutines take *planes* as an argument. *planes* is an integer mask that specifies which planes of the display are to be modified by a display function (*func*), one bit per plane. A monochrome display has only one plane and is the least-significant bit in the mask. As planes are added, they occupy more significant bits. You can use the constant **AllPlanes** to specify all planes.

## Brush

A *brush* is a rectangular area that is painted in the line-drawing subroutines at each point of the line or curve. The upper left corner of the brush follows the line. If the width or height of the brush is greater than one pixel, the display hardware may paint some pixels more than once. Therefore, you should not use brushes with display functions that are invertible. " Display Functions" on page 4-24 defines the invertible display functions.

## Clip Mask

A *clip mask* is a bitmap used to specify which pixels in the destination can be modified. If a bit in the clip mask is set to 1, the destination can be modified; if the clip mask bit is 0, the destination cannot be modified. The clip mask must be exactly the same height and width as the destination region.

A clip mask may be useful for icon generation.

## Color

On color displays, the X Server keeps currently-defined colors in a color map. A color map is an array of color cells. Each color cell is made up of three intensity values: red, green, and blue.

The size of the color map (number of color cells) determines the number of colors that can be displayed simultaneously. Use the **DisplayCells( )** macro to get the size of the color map.

A color is an index into the color map. A pixel value is a color. For each possible value a pixel may take there is a color cell in the color map. For example, if a display is four planes deep, pixel values 0 through 15 are defined.

The display hardware takes each pixel value in display memory and uses it as an index into the color map. The red, green, and blue values in the cell determine the color displayed on the screen.

The following structure is used to define a color. It can be passed to or from the X Server in various color function calls and is defined in **X/X.h.**

```
typedef struct _ColorDef
  unsigned short pixel;                    /* index into color map */
  unsigned short red, green, blue;         /* RGB intensity values */
  ColorDef;
```

The red, green, and blue values are scaled. Full intensity is 0xFFFF, half intensity is 0x8000 and off is 0x0000. This representation gives uniform results across displays with different numbers of planes.

The first two pixel values are predefined as black (0) and white (1). The constants **BlackPixel** and **WhitePixel** can be used.

On monochrome displays, only **BlackPixel** and **WhitePixel** are defined. If other pixel values are detected by the X Server, they are converted to **BlackPixel** (even values) or **WhitePixel** (odd values).

Be careful when using display functions other than **GXcopy** on a color display. Display functions such as **GXxor** and **GXinvert** can result in undefined pixel values. For example, if an X Server on a four-plane display performs the **GXinvert** on pixel value 3, the result is pixel value 12. The color currently associated with pixel value 12 is displayed, whether or not you ever defined pixel value 12.

## Font

A font as used in X-Windows is described by a **FontInfo** structure that contains various fields describing the font. This structure is defined as follows:

**typedef struct _FontInfo {**
    **Font** *id*;
    **short** *height*;
    **short** *width*;
    **short** *baseline*;
    **short** *fixedwidth*;
    **short** *firstchar*;
    **short** *lastchar*;
    **short** *\*widths*;
**} FontInfo;**

Fields in the structure are defined as follows:

| | |
|---|---|
| *id* | ID of the font. |
| *height* | Constant value for all characters in the font. |
| *width* | Average width of characters in the font. |
| *baseline* | Baseline of characters in the font. This value indicates the number of pixels from the bottom of the font that characters without descenders begin. |
| *fixedwidth* | Set to 1 if fixed width, 0 if variable width. A font is fixed-width if all characters in the range of legal characters are the same width. |
| *firstchar, lastchar* | First and last characters in the font, respectively. |
| *widths* | Pointer to the font width array. |

## Cursors and Locators

From the X Window perspective, a cursor consists of a cursor shape, mask, colors for the shape and mask, a *hot spot*, and logical function. The hot spot defines the point on the cursor that is reported when a mouse event occurs. The cursor bitmap determines the shape of the cursor. The mask bitmap determines the bits that are modified by the cursor. The colors determine the colors of the shape and mask. The **GXcopy** and **GXxor** display functions are supported. Use **XQueryCursorShape** to determine the range of possible cursor sizes.

Whenever the mouse cursor is in a visible window, it is set to the cursor defined for that window. If no cursor is defined for that window, the cursor is set to that of the parent window.

A locator is an absolute point on a window represented as $(x, y)$. The x coordinate is contained in the most significant 16 bits, and the y coordinate is contained in the least significant 16 bits.

## Speaker Volume

The volume of sounds emitted by the speaker when in use by X-Windows is controlled by the **XKeyClickControl**, **XFeep** and **XFeepControl** functions. The RT PC supports volume levels. These levels map to the numeric values used by the X-Windows functions as follows:

| Volume | Value |
|--------|-------|
| Off | 0 |
| Low | 1, 2, 3 |
| Medium | -1, 4, 5, 6 |
| High | 7, 8 |

## Draw Operations

Polygon draw operations are accomplished by a list of points that are connected by lines. The points, or vertexes, are defined in the following structure supplied by the caller:

**typedef struct _Vertex {**
 **short** *x, y*;
 **unsigned short** *flags*;
  **} Vertex;**

*x* and *y* are the coordinates of the vertex, relative to either the upper left inside corner of the window (if the **VertexRelative** bit in the flags definition is 0) or the previous vertex (if **VertexRelative** is 1).

The flags, as defined in < X/X.h >, are defined as follows:

| Flag | Value | Meaning |
|------|-------|---------|
| VertexRelative | 0x0001 | Else vertex absolute |
| VertexDontDraw | 0x0002 | Else vertex draw |
| VertexCurved | 0x0004 | Else straight |
| VertexStartClosed | 0x0008 | Else not start of closed curve |
| VertexEndClosed | 0x0010 | Else not end of closed curve |
| VertexDrawLastPoint | 0x0020 | Else don't draw last point |

If bit 0 of **VertexRelative** is not set, the coordinates are absolute (relative to the window). The first vertex must be an absolute vertex.

If the **VertexDontDraw** bit is 1, no line or curve is drawn from the previous vertex to this one. This is analogous to picking up the pen and moving to another place before drawing another line.

If the **VertexCurved** bit is 1, a spline algorithm is used to draw a smooth curve from the previous vertex, through this one, to the next vertex. Otherwise, a straight line is drawn from the previous vertex. Set **VertexCurved** to 1 only if a previous and next vertex are both defined (either explicitly in the array, or through the definition of a closed curve).

**VertexDontDraw** bits and **VertexCurved** bits can both be 1; this is useful if you want to define the previous point for the smooth curve, but do not want an actual curve drawing to start until this point.

If the **VertexStartClosed** bit is 1, then this point marks the beginning of a closed curve. This vertex must be followed later in the array by another vertex whose absolute coordinates are identical, and which has a **VertexEndClosed** bit of 1. The points in between form a cycle for the purpose of determining predecessor and successor vertices for the spline algorithm. Set the **VertexStartClosed** bit or the **VertexEndClosed** bit to 1 only if **VertexCurved** is also 1.

Normally, the end point of a curve or line is not drawn, since it is probably the beginning point of the next curve or line. This is important if a display function such as **GXinvert** or **GXxor** is used, since drawing a point twice with such a function produces a different result than drawing it just once. If **VertexDrawLastPoint** is 1, the end point is drawn.

## Association Tables

Because application programs often need to refer to their own data structures when an event arrives, an **XAssocTable** can be used to classify X resources. Suppose you want to have three or four types of windows, each type with different properties. You can associate each window ID with a pointer to a window properties structure you have defined. A generic association table for resource IDs has been defined in the X library, under the name **XId**.

Observe the following guidelines when using an **XAssocTable**:

- All **XId**s are relative to the currently active display. Therefore, if you are using multiple displays, you must ensure the proper display is active before performing an **XAssocTable** operation. **XAssocTable** imposes no limitations on the number of **XId**s per table, the number of **XId**s per display, or the number of displays per table.

- The hashing scheme used by the association mechanism functions more efficiently when the table size (the number of buffers in the hashing system) is a power of two and if there are no more than eight **XId**s per buffer.

# Tiles

A tile is 16 bits wide by 16 bits high. Background tiling uses Pixmaps to specify the pattern. Because patterns can be absolute or relative to the window, a tile mode is associated with a window. The tile mode allows a pattern to be positioned absolute to the window or relative to the parent window (often the root window). Thus, patterns can be aligned either to the window in which you are working or to the parent window. Both modes are useful, as icon windows often need relative alignment and normal windows prefer absolute alignment.

# Macros and Constants

The following macros and definitions are found in **X/X.h**:

| | |
|---|---|
| **BitmapSize(***width, height***)** | This macro computes bitmap sizes in bytes. |
| **XYPixmapSize(***width, height, planes***)** | This macro computes XY format pixmap sizes in bytes. |
| **WZPixmapSize(***width, height***)** | This macro computes Z format pixmap sizes in bytes. |
| **BZPixmapSize(***width, height***)** | This macro computes Z format pixmap sizes in bytes. |
| **BlackPixel** | The pixel value 0 (may not actually be black). |
| **WhitePixel** | The pixel value 1 (may not actually be white). |

The following macros and definitions are found in **X/Xlib.h**:

**AllPlanes** − value for plane mask when all planes are used (~0).
**DashedLine** − dashed line pattern
**DottedLine** − dotted line pattern.
**DotDashedLine** − dot/dashed line pattern.
**SolidLine** − solid line pattern.

The following macros must not be used before **XOpenDisplay** is called:

**RootWindow** − window ID of the root window.
**BlackPixmap** − pixmap ID of a black pixmap (may not actually be black).

**WhitePixmap** − pixmap ID of a white pixmap (may not actually be white).
**ProtocolVersion( )** − X protocol version number.
**dpyno( )** − display file descriptor for use with **select( )**.
**DisplayType( )** − display type.
**DisplayPlanes( )** − number of display planes.
**DisplayCells( )** − number of color cells supported by the display.
**DisplayName( )** − name of the display.

For a description of macros added by IBM, see "IBM-Specific X-Windows Implementation" on page .

# Subroutine Reference Information

The rest of this chapter contains more detailed descriptions of the X-Windows subroutines. The subroutines are arranged in alphabetical order.

Many functions return an integer resource ID. These can be of type **Window**, **Font**, **Pixmap**, **Bitmap**, or **Cursor** (as defined in **/usr/include/X/X.h**). Some functions return **Status**, which is an integer error code. If a function fails, it returns 0.

If a client does not want a request to execute asynchronously, it should be followed immediately by a call to **XSync**, which blocks until all previously-buffered asynchronous events have been sent and acted upon.

## DisplayHeight

**DisplayHeight ( )**

This function returns the height of the physical display in pixels.

## DisplayWidth

**DisplayWidth ( )**

This function returns the width of the physical display in pixels.

## XAddHost

**#include < sys/socket.h >**

**XAddHost (*host*)**
**struct innadr \****host***;

*host*        The network address of the host to add.

The add host function adds the specified host to the list of hosts that are allowed to open connections to this display. The program and the display hardware must be on the same host.

# XAppendToBuffer

**XAppendToBuffer** (*bytes, nbytes, buffer*)
**char \****bytes*;
**int** *nbytes*;
**int** *buffer*;

*bytes*      Pointer to the bytes to append.

*nbytes*     Number of bytes to append.

*buffer*     Buffer of bytes to append.

This function appends an arbitrary string of bytes onto the specified buffer. The contents of the buffer may be retrieved later by any client that calls **XFetchBytes**. Note that the buffer's contents are not necessarily ASCII or null-terminated, so null bytes are not special.

# XAppendVertex

**int XAppendVertex** (*vertices, nvert*)
**Vertex** *vertices*[ ];
**int** *nvert*;

*vertices*   Vertices to append.

*nvert*      Indicates successful append.

The append vertex function appends vertices to the output buffer. This function is not intended for most users of the library, but is intended as a hook for certain libraries built on this library.

Upon successful completion, this function returns *nvert*. If there was no **Draw** command currently in the output buffer, the function returns 0. If the vertices would not fit before the end of the buffer was reached, -1 is returned.

# XAutoRepeat

**XAutoRepeatOn ()**

**XAutoRepeatOff ()**

**XAutoRepeatOn** turns the keyboard auto-repeat function on. **XAutoRepeatOff** turns the keyboard put-repeat function off.

# XBitmapBitsPut

**XBitmapBitsPut (***w, x, y, width, height, data, foreground,*
                    *backgrount, clipmask, func, planes***)**
**window** *w*;
**short \****data*;
**bitmap** *clipmask*;
**int** *func, planes*;

| | |
|---|---|
| *w* | The window that contains the region. |
| *x, y* | The coordinates of the top left corner of the region relative to the top left corner of the window. |
| *width* | The width of the region. |
| *height* | The height of the region. |
| *data* | Pointer to the data area to contain the bitmap. |
| *foreground* | The foreground pixel value. |
| *background* | The background pixel value. |
| *clipmask* | Clip mask. |
| *func* | Display function. |
| *planes* | Plane mask. |

The put bitmap routine performs a display function in a region of a window using a pixmap defined by a bitmap and a pair of source pixels that define the foreground and background pixel values. See "Bitmaps" on page 4-15 for more information.

## XChangeBackground

**XChangeBackground** (*w*, *tile*)
**window** *w*;
**Pixmap** *tile*;

*w*        The window to be retiled.

*tile*      The Pixmap to be used for the retile operation.

The change background function changes the background *tile* of the specified window. If *tile* is not specified, the background pixmap of the parent window is used or, if it is the root window, the default background is restored. You can free *tile* immediately if no further explicit references to it are made. This function is only valid on an opaque window and an error results if *w* is transparent.

**Note:** This function does not change the current contents of the window. It simply resets the background pixmap to be used when the background is refreshed.

## XChangeBorder

**XChangeBorder** (*w*, *tile*)
**window** *w*;
**Pixmap** *tile*;

*w*        The window to be changed.

*tile*      The source Pixmap for the change.

The change border function changes and repaints the border of the window. You can immediately free *tile* if no further explicit references are made to it.

This function is only valid on an opaque window with a border. An error results if the window is not opaque or if the window is opaque but does not have a border.

## XChangeWindow

**XChangeWindow** (*w*, *width*, *height*)
**window** *w*;
**int** *width*, *height*;

*w*        The window to be changed.

| | |
|---|---|
| *height* | New height of the window from the upper-left coordinate, not including the border. |
| *width* | New width of the window from the upper-left coordinate, not including the border. |

This function changes the size of the specified window without changing its upper-left coordinate. **XChangeWindow** always raises the window.

Changing the size of a mapped opaque window loses its contents and generates an **ExposeWindow** event. Changing the size of a transparent window does not affect the screen.

If a mapped opaque window is made smaller, exposure events are generated on opaque windows that it formerly obscured. The origin of the window is not changed.

# XCharBitmap

**Bitmap XCharBitmap** (*font, char*)
**Font** *font.*;
**int** *char*;


| | |
|---|---|
| *font* | Font ID. |
| *char* | Character from the specified font for which to create a bitmap. |

The character bitmap function creates a bitmap of a single character from the specified font. You can free the font if no further references to it are to be made.

# XCharWidths

**Status  XCharWidths** (*chars, len, fonts, widths*);
**char** *\*chars*;
**int** *len*;
**Font** *font*;
**short** *\*widths*;


| | |
|---|---|
| *chars* | Pointer to a string of characters, not necessarily null-terminated, for which the width of each character is provided. |
| *len* | The number of characters in the string *chars*. |
| *font* | The font from which the character string is obtained. All characters in a string must be from the same font. |

*widths*     Pointer to an array containing the width of each character in the supplied string. For example, *widths[3]* is set to the width of the character *chars[3]*.

The character width function determines the width of each character in a supplied string and stores the width in an array that has as many entries as characters in the supplied string. The width value is expressed in pixels.

# XCheckMaskEvent

int **XCheckMaskEvent** (*mask, rep*)
int *mask*;
**XEvent** *\*rep*;

*mask*     Mask that identifies an event to check for in a queue.

*rep*     Pointer to an **XEvent**.

The check mask event function searches a queue for the event identified by *mask*. First, **XCheckMaskEvent** flushes the output buffer, and searches the queue for an event that matches the passed mask. If the event is found in the queue, this function returns a value of 1 and copies the event into an **XEvent** supplied by the caller. Events earlier in the queue are not discarded.  If no such event has been queued, **XCheckMaskEvent** immediately returns 0.

# XCheckWindowEvent

int **XCheckWindowEvent** (*w, mask, rep*)
**Window** *w*;
int *mask*;
**XEvent** *\*rep*;

*w*     Window to check for in the queue.

*mask*     Mask that identifies the event to search for in the queue.

*rep*     Pointer to an **XEvent**.

The check window event function searches the input queue for an event that matches both the passed window and the passed mask.  First, **XCheckWindowEvent** flushes the output buffer, and searches the queue for an event that matches the passed window and passed mask.  If the event is found in the queue, this function returns a value of 1 and copies the event into an **XEvent** supplied by the caller. Events earlier in the queue are not discarded. If no such event has been queued, **XCheckWindowEvent** immediately returns 0.

## XCircWindowDown

**XCircWindowDown** (*w*)
**Window** *w*;

*w*          Window whose child is to be lowered.

This function lowers the highest-mapped child of the window that partially or completely obscures another child. Completely unobscured children are not affected. This function generates exposure events on any window formerly obscured, and repeated executions lead to round robin lowering.

## XCircWindowUp

**XCircWindowUp** (*w*)
**Window** *w*;

*w*          Window whose child is to be raised.

This function raises the lowest-mapped child of the window that is partially or completely obscured by another child. Completely unobscured children are not affected. This function generates exposure events on that child (and its opaque descendants) if any part of it was formerly obscured. Repeated executions lead to round robin raising.

## XClear

**XClear** (*w*)
**Window** *w*;

*w*          Window to be cleared and repainted.

The clear function clears the specified window and repaints it with the background. If the window is transparent, it is cleared and repainted with its parent's background. **XClear** never generates exposure events.

## XCompressEvents

**XCompressEvents ( )**

This function suppresses all but the last **MouseMoved** event if multiple **MouseMoved** events have been received without any other intervening events or replies. You can call **XExpandEvents** to report all events. By default, all **MouseMoved** events are compressed.

## XCondWarpMouse

**XCondWarpMouse** (*dw, dx, dy, sw, sx, sy, swidth, sheight*)
**Window** *dw, sw*;
**int** *dx, dy, sx, sy*;
**int** *swidth, sheight*;

*dw, sw*        Destination window and source window, respectively.

*dx, dy, sx, sy* Coordinates of the destination and source windows, respectively.

*swidth, sheight* Width and height, respectively, of the source window.

This function moves the mouse to the destination position relative to the origin of the destination window (*dw*), but only if the mouse is currently in a visible region of the specified region (*sx, sy, swidth, sheight*) of the source window (*sw*).

If the source height is zero, the current height (*sheight*) of the source window minus the source right (*sy*) coordinate is used. If the source width (*swidth*) is zero, the current width of the source window minus the source left (*sx*) coordinate is used.

## XConfigureWindow

**XConfigureWindow** (*w, x, y, width, height*)
**Window** *w*;
**int** *x, y, width, height*;

*w*            Window to configure.

*x, y*         Coordinates of the window to configure.

*width, height* Width and height of the window to configure.

**XConfigureWindow** changes the size and location of the specified window. Configuring a mapped opaque window loses its contents and generates an ExposeWindow event;

configuring a transparent window does not affect the screen. **XConfigureWindow** always raises the window.

Configuring a window may generate exposure events on opaque windows that the window formerly obscured, depending on the new size and location parameters.

## XCopyArea

**XCopyArea** (*w, sx, sy, dx, dy, width, height, func, planes*)
**Window** *w*;
**int** *sx, sy, dx, dy, width, height*;
**int** *planes*;
**int** *func*;

| | |
|---|---|
| *w* | Window in which the copy is to be made. |
| *sx, sy* | Coordinates of the area to be copied. |
| *dx, dy* | Coordinates of the location at which to copy the specified area. |
| *width, height* | Width and height of the area to be copied. |
| *func* | Display function. |
| *planes* | Plane mask. |

**XCopyArea** copies one region of the window to another (possibly overlapping) region of the same window, using the supplied display function *func*.

If parts of the source region are obscured, the corresponding parts of the destination are painted with the background tile. If a client has called **XSelectInput** on this window with the **ExposeCopy** bit set, then **ExposeRegion** events are generated on any such parts of the destination, and then an **ExposeCopy** event is generated. All of these events are guaranteed to be together in the stream, with no intervening events. This sequence makes it possible to scroll the contents of a window, getting exposure events from wherever the window was obscured to refresh those areas of the screen.

## XCreate

**XCreate** (*prompt, program, geometry, default, frame, minw, minh*)
**char** *\*prompt*;
**char** *\*program*;
**char** *\*geometry, \*default*;
**OpaqueFrame** *\*frame*;
**int** *minw, minh*;

*prompt*    Character string used in a prompt window to inform the user of the application to be placed in the window.

*program*    Name to be used by **XGetDefault** to get user defaults.

*geometry, default* Specify the placement and/or size of the window.

*frame*    Specifies the background and border pixmaps, as well as the window's border width.

*minw*    Minimum width of the window in pixels.

*minh*    Minimum height of the window in pixels.

**XCreate** does all the work for automatic and manual placement of a window, and is commonly used by most applications for creating graphics-related windows.

The *minw* and *minh* specify the minimum size of the created window in pixels.

The *prompt* argument is used in a prompt window (if needed) to inform the user what application wants to be placed. The *program* name must be passed in with the program argument (usually it should be **argv[0]**) so that **XGetDefault** can find out how the user likes to be prompted to create the window.

The *geometry* and *default* arguments are used to place the position and/or size of the window.

The *frame* passed in must include the background and border pixmaps already specified, and the border width of the window.

The window is not mapped after creation. The function returns the window ID of the window just created, and all values in the passed-in window *frame* is set.

For more information on window creation, see "Creating Windows" on page 4-13.

# XCreateAssocTable

**XCreateAssocTable \*XCreateAssocTable (***size***)**
**int** *size*

*size*    The number of buffers used by **XAssocTable**.

The create association table function returns a pointer to a newly created **XAssocTable**. For efficiency, the *size* parameter (number of buffers) should be a power of 2, with a maximum of 8 objects per buffer. A null pointer is returned if there is an error allocating memory.

# XCreateCursor

## XCreateCursor

Cursor **XCreateCursor** (*width, height, cursor, mask, xoff, yoff, fg, bg, func*)
int *width, height*;
short \**cursor*, \**mask*;
int *xoff, yoff*;
int *fg, bg*;
int *func*;

*width, height* Width and height of the desired cursor.

*cursor, mask* Pointers to areas in bitmap format that define the cursor and mask.

*xoff, yoff*    Coordinates of a point in the bitmap that corresponds to the position of the mouse.

*fg*    Foreground color of the cursor.

*bg*    Background color of the cursor.

*func*    Display function (**GXcopy** or **GXxor**).

This function Creates a cursor out of its component parts from data in the calling program. The *cursor* bits and *mask* bits should be in bitmap format. This function is used if all components of a cursor are in the client program, and saves time in defining the cursor.

## XCreateTerm

Window **XCreateTerm** (*prompt, program, geometry, default, frame, minw, minh, xadder, yaddr, cwidth, cheight, font, fwidth, fheight*)
char \**prompt*;
char \**program*;
char \**geometry*, \**default*;
**OpaqueFrame** \**frame*;
int *minw, minh*;
int *xaddr, yaddr*;
int \**cwidth*, \**cheight*;
**FontInfo** \**font*;
int *fwidth, fheight*;

*prompt*    Character string used in a prompt window to inform the user of the application to be placed in the window.

*program*    Name of the program to be used by **XGetDefault** for prompting the user to create a window.

## XClearIconWindow

**XClearIconWindow** (*w*)
**Window** *w*;

*w*          Window for which to clear the icon window.

**XClearIconWindow** clears the icon window specification for a window. See "XSetIconWindow" on page 4-92 for more information.

## XClearVertexFlag

**XClearVertexFlag ( )**

This macro clears the state of the flag that indicates if it is safe to append vertices. This macro is intended principally for use by more sophisticated libraries.

## XClipClipped

**XClipClipped** (*w*)
**Window** *w*;

*w*          Window for which to set clip mode.

This function, along with **XClipDrawThrough**, determines what happens when you draw on part of window obscured by a child window. If **XClipClipped** is in effect, output into areas covered by children is suppressed. All windows start out in clipped mode when created by **XCreatewindow**.

## XClipDrawThrough

**XClipDrawThrough** (*w*)
**Window** *w*;

*w*          Window for which to set draw-through mode.

This function, along with **XClipClipped**, determines what happens when you draw on part of window obscured by a child window. If the clip mode is **DrawThrough**, output is drawn on the screen as if the child window was not there. **DrawThrough** is useful for drawing window outlines when moving or resizing windows. Note that any such requests must be

atomic and return the screen to its original state in a single X request. The root window's clip mode is initially **DrawThrough**.

## XCloseDisplay

**XCloseDisplay** (*display*)
**Display** *\*display*;

*display*    Pointer to the display structure returned by **XOpenDisplay**.

**XCloseDisplay** closes the connection associated with the specified *display*. All windows or other resources that the caller has created on this display server are destroyed; they should never be referenced again. Any output events that have been buffered but not yet sent are discarded.

The effect of **XCloseDisplay** is automatically achieved if a process exits. For this reason, most clients do not need to call **XCloseDisplay**.

If a client has created Window, Font, Bitmap, Pixmap, or Cursor resource ID's with this display server, they must not be used after calling **XCloseDisplay**.

**Note:** If your program has a connection to a display server open, then issues the **fork** system call, you must take care when dealing with the connection to the window system. Do not forget to flush the output buffer before waiting on the child process, and do not forget to make sure that you have processed all input events before forking or exiting in the child, or your programs may perform the operations twice.

## XCloseFont

**XCloseFont** (*info*)
**FontInfo** *\*info*;

*info*    Pointer font information structure.

This function closes off any use of a font, and deallocates the storage associated with the **FontInfo** structure.

**Note:** Do not use **XCloseFont** to close a font not opened with **XOpenFont**, because you may corrupt the memory pool.

*geometry, default* Specify the placement and/or size of the window.

*frame* Specifies the background and border pixmaps, as well as the window's border width.

*minw* Minimum width of the window in multiples of *fwidth*.

*minh* Minimum height of the window in multiples of *fheight*.

*xaddr, yaddr* Provides additional interior padding needed in the window.

*cwidth, cheight* Pointers to the returned width and height, respectively, of the created window.

*font* Font ID.

*fwidth, fheight* Specify the size of the units used in the geometry specification and the increments the the window is sized in.

**XCreateTerm** does all the work for automatic and manual placement of a window to be sized in multiples of *fwidth* and *fheight* and is commonly used by most applications for creating text-related windows.

The *prompt* argument is used in a prompt window (if needed) to inform the user what application wants to be placed. The *program* name must be passed-in with the program argument (usually it should be **argv[0]**) so that **XGetDefault** can find out how the user likes to be prompted to create the window.

The *geometry* and *default* arguments are used to place the position and/or size of the window.

The *fwidth* and *fheight* arguments specify the size of the units used in the geometry spec, and the increments the window should be sized in. These are typically the size of a fixed-width font, or other graphic object in a window.

The *frame* passed in must include the background and border pixmaps already specified, and the border width of the window. *xaddr* and *yaddr* provide additional interior padding needed in the window. The function returns the ID of the window just created, and all values in the passed-in window frame are set.

The window is not mapped after creation.

For more information on window creation, see "Creating Windows" on page 4-13.

## XCreateTransparencies

**Window XCreateTransparencies** (*parent, defs, ndefs*)
**Window** *parent*;
**TransparentFrame** *defs*[ ];
**int** *ndefs*;

*parent*    Parent window parameter.

*defs*      Array of window information definitions.

*ndefs*     Number of window definitions in the array.

**XCreateTransparencies** takes an array of window information definitions and creates the transparnecies in a single operation with the window system. The caller should fill in the structure (except for the window IDs). The IDs of the created windows are returned in the structure passed to the subroutine. You must specify the number of windows to be created using the *ndefs* parameter. This function returns the the number of windows actually created, or 0 if unsuccessful.

## XCreateTransparency

**Window XCreateTransparency** (*parent, x, y, width, height*)
**Window** *parent*;
**int** *x, y, width, height*;

*parent*    Parent window parameter.

*x, y*      Coordinates of the window relative to the parent window.

*width, height* Width and height of the newly-created window.

**XCreateTransparency** creates an unmapped transparent window. Transparent windows do not have borders. The coordinates are relative to the parents coordinate system. The window does not initially have a cursor registered and has a clipmode of **ClipModeClipped** (output to the window is obscured by subwindows of the parent). A transparent window does not have a background pattern. It initially has a tile mode of **TileModeRelative**.

The subroutine returns the window ID of the created window or 0 if the subroutine fails.

# XCreateWindow

Window **XCreateWindow** (*parent, x, y, width, height, brdrwidth, brdr, bg*)
**Window** *parent*;
**int** *x, y, width, height*;
**int** *brdrwidth*;
**Pixmap** *brdr*;
**Pixmap** *bg*;

| | |
|---|---|
| *parent* | Window from which the subwindow is created. |
| *x,y* | Coordinates of the top left outside corner of the new window. |
| *width, height* | Indicate the inside dimensions of the new window, not including the borders. |
| *brdrwidth* | Width in pixels of the new window's borders. |
| *brdr* | Pixmap of the border, if the border width is greater than zero. |
| *bg* | Pixmap of the background; if not specified, the parent window's pixmap is used instead. |

**XCreateWindow** creates an unmapped, opaque subwindow of the specified parent window, which must also be opaque. The created window is always wholly contained within its *parent*; any part of the window extending outside its parent window is clipped.

The *x* and *y* coordinates represent the top left outside corner of the new window's borders. They are relative to the inside of the *parent* window's borders. The *width* and *height* parameters are the new window's inside dimensions--they do not include the new window's borders, which are entirely outside of the window. The new window has a border *brdrwidth* pixels wide.

A *brdr* Pixmap need not be given if the border width is zero, in which case the window does not have a border. If no *bg* Pixmap is given, the parent's background Pixmap is used. A monochrome application may find the macros **BlackPixmap** and **WhitePixmap** useful; these refer to solid black and white pixmaps that are automatically created by **XOpenDisplay**.

The **tilemode** of the new window is absolute. The **clipmode** of the new window is clipped. The new window does not have an associated icon window. The **name** of the window is the null string.

The created window is not yet mapped to the user's display; to display the window, call **XMapWindow**. The new window does not have a cursor defined; the cursor is that of the window's parent unless a new cursor is registered. The window is not visible on the screen unless it and all of its ancestors are mapped, and it is not obscured by any of its ancestors.

This function returns the window ID of the created window, or 0 if the subroutine fails.

## XCreateWindowBatch

> int **XCreateWindowBatch** (*parent, defs, ndefs*)
> **Windowto** *parent*;
> **BatchFrame** *defs*[];
> int *ndefs*;

*parent*   Parent window from which subwindows are to be created.

*defs[]*   Array of window definitions.

*ndefs*   Number of windows to be created.

This function takes an array of window information definitions and creates the windows of the specified types in a single handshake with the window system. The caller should have filled in the structure except for the window ID. The window IDs of the created windows are returned in the structure passed to the subroutine. You must specify the number of windows to be created using the *ndefs* argument. The other results are the same as for **XCreateWindow** and **XCreateTransparency**. The subroutine returns the number of windows actually created. The parent windows must already exist.


## XCreateWindows

> int **XCreateWindows** (*parent, defs, ndefs*)
> **Window** *parent*;
> **OpaqueFrame** *defs*[];
> int *ndefs*;

*parent*   Parent window from which subwindows are created.

*defs[]*   Array of opaque window definitions.

*ndefs*   Number of windows to be created.

This subroutine takes an array of window information definitions and creates them in a single handshake with the window system. The caller should have filled in the structure except for the window ID. The window IDs of the created windows are returned in the structure passed to the subroutine. You must specify the number of windows to be created using the *ndefs* argument. The other results are the same as for **XCreateWindow**. The subroutine returns the number of windows actually created.

## XDefineCursor

**XDefineCursor** (*w, cursor*)
**Window** *w*;
**Cursor** *cursor*;

*w*        The window for which a cursor is being defined.

*cursor*   Cursor ID.

If a cursor is specified, it is used when the mouse is in the window.

## XDeleteAssoc

**XDeleteAssoc** (*table, xid*)
**XAssocTable** *\*table*;
**XId** *xid*;

*table*    Pointer to an association table.

*xid*      X resource ID.

This function deletes an association in an **XAssocTable** keyed on its XId. Redundant deletes (and deletes of unknown XId's) are meaningless and cause no problems. Deleting associations in no way impares the performance of an **XAssocTable**.

## XDestroyAssocTable

**XDestroyAssocTable** (*table*)
**XAssocTable** *\*table*;

*table*    Pointer to an association table.

This function frees the memory associated with an association table.

**Note:** Do not use an association table after it has been destroyed.

## XDestroySubwindows

**XDestroySubwindows** (*w*)
**Window** *w*;

w        All subwindows of this window are destroyed.

**XDestroySubwindows** destroys all subwindows of this window. The subwindows should never again be referenced.

**XDestroySubwindows** generates exposure events on **w**, if any mapped opaque subwindows were actually destroyed.

Use this function to delete many windows (rather than deleting them one at a time), as much of the work need only be performed once for all of the windows rather than for each window.

## XDestroyWindow

**XDestroyWindow** (*w*)
**Window** *w*;

w        Window to be unmapped and destroyed.

**XDestroyWindow** unmaps and destroys the window and all of its subwindows. The windows should never again be referenced. **XDestroyWindow** may be called on either opaque or transparent windows.

If the window has an icon window, that icon window is automatically unmapped and sent an unmap window event. If the window is a mapped icon window, its corresponding window is automatically mapped. This prevents windows being lost when a window manager exits unexpectedly.

Destroying a mapped opaque window generates exposure events on other opaque windows obscured by the window being destroyed.

**XCloseDisplay** automatically destroys all windows that have been created on that server (unless called after a **fork()**). See **Note** under **XCloseDisplay**.

# XDisplayName

**XDisplayName** (*display*)
**Display \****display***;

*display*     Pointer to the Display structure returned by **XOpenDisplay**.

**XDisplayName** returns the name of the display **XOpenDisplay** would use. This is useful for printing the name of the display if **XOpenDisplay** returns an error and the Display variable is NULL.

# XDraw

**XDraw** (*w, vlist, vcount, width, height, pixel, func, planes*)
**Window** *w*;
**Vertex \****vlist***;
**int** *vcount*;
**int** *height, width*;
**int** *pixel*;
**int** *func*;
**int** *planes*;

*w*            Window in which to draw.

*vlist*        List of vertices used to draw the polygon or curve.

*vcount*       Number of vertices in *vlist*.

*height, width* Size of the brush to draw along the outline of the figure.

*pixel*        Source pixel value (color).

*func*         Display function.

*planes*       Plane mask.

**XDraw** draws an arbitrary polygon or curve using the specified function. The figure drawn is defined by the list of Vertexes *vlist* that the caller supplies. See "Draw Operations" on page 4-28 for the format of the structure. The points are connected by lines as specified by the flags in the vertex structure.

*width* and *height* specify the size of a brush to be drawn along the line.

The line is drawn in the color specified by the pixel value.

## XDrawDashed

**XDrawDashed** (*w, vlist, vcount, width, height, pixel, pattern, func, planes*)
**Window** *w*;
**Vertex \****vlist*;
**int** *vcount*;
**int** *width, height*;
**int** *pixel*;
**Pattern** *pattern*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Window in which to draw. |
| *vlist* | List of vertices used to draw the polygon or curve. |
| *vcount* | Number of vertices in *vlist*. |
| *width, height* | Size of the brush to draw along the outline of the figure. |
| *pixel* | Source pixel value (color). |
| *pattern* | Value encoding the pattern to use for the line. |
| *func* | Display function. |
| *planes* | Plane mask. |

This function is similar to **XDraw**, except that it draws a dashed rather than a solid line. The pattern is a value encoding the pattern to be used. (See **XMakePattern** for more details.) The destination is only updated when the pattern bit is one.

## XDrawFilled

**XDrawFilled** (*w, vlist, vcount, pixel, func, planes*);
**Window** *w*;
**Vertex \****vlist*;
**int** *vcount*;
**int** *pixel*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Window in which to draw. |
| *vlist* | List of vertices used to draw the polygon or curve. |
| *vcount* | Number of vertices in *vlist*. |

*pixel*  Value to use to fill in the figure.

*func*  Display function.

*planes*  Plane mask.

**XDrawFilled** draws arbitrary polygons or curves and fills them with the specified **pixel** value. The vertex list should consist only of one or more closed regions. A point is defined to be inside a region if an infinite ray with the point as an origin crosses the path of the region an odd number of times.

# XDrawPatterned

**XDrawPatterned** (*w, vlist, vcount, width, height, pixel, altpix, pattern, func, planes*)
**Window** *w*;
**Vertex** *\*vlist*;
**int** *vcount*;
**int** *width, height*;
**int** *pixel*;
**int** *altpix*;
**Pattern** *pattern*;
**int** *func*;
**int** *planes*;

*w*  Window in which to draw.

*vlist*  List of vertices used to draw the polygon or curve.

*vcount*  Number of vertices in *vlist*.

*width, height* Size of the brush to draw along the outline of the figure.

*pixel*  Source pixel value (color) when pattern bit is 1.

*altpix*  Alternate source pixel value when the pattern bit is 0.

*pattern*  Value encoding the pattern to use for the line.

*func*  Display function.

*planes*  Plane mask.

This function is similar to **XDraw**, except that it draws a patterned rather than a solid line. The pattern is a value encoding the pattern to be used. For a patterned line, the source pixel value is used when the pattern bit is 1 and the alternate source pixel value is used when the pattern bit is 0.

## XDrawTiled

**XDrawTiled** (*w, vlist, vcount, tile, func, planes*)
**Window** *w*;
**Vertex** *\*vlist*;
**int** *vcount*;
**int** *tile*;
**int** *func*;
**int** *planes*;

*w*       Window in which to draw.

*vlist*    List of vertices used to draw the polygon or curve.

*vcount*   Number of vertices in *vlist*.

*tile*     Tile pixmap to use to fill the figure.

*func*     Display function.

*planes*   Plane mask.

**XDrawTiled** draws arbitrary polygons or curves and fills them with the specified *tile*
Pixmap. Note that there may be implementation restrictions on the nature of the *tile*
Pixmap. (See **XQueryTileShape**.) The vertex list should consist only of one or more
closed regions. A point is defined to be inside a region if an infinite ray with the point as
an origin crosses the path of the region an odd number of times.

## XErrDescrip

**char \*XErrDescrip** (*code*)
**int** *code*;

*code*     Error code.

This routine returns a null-terminated string describing the specified error code. The
string is static in **Xlib** and should not be modified or freed. For a list of error codes, see
**XErrorHandler**.

# XErrorHandler

**XErrorHandler (***handler***)**
**int** *handler(Display \*, XErrorEvent \*)*;

*handler*    The error handler supplied by the program.

The program's supplied error handler is called by **Xlib** whenever an **XErrorEvent** is received. This is not assumed to be a fatal condition (it is acceptable for this procedure to return). However, the error handler should not perform any operations (directly or indirectly) on the Display. The fields of the **XErrorEvent** passed to **XErrorHandler** should be interpreted as follows:

```
typedef struct _XErrorEvent {
        long pad;
        long serial;                    /* serial number of failed request */
        char error_code;                /* error code of failed request */
        char request_code;              /* request code of failed request */
        char func;                      /* function field of failed request */
        char pad_b7;
        Window window;                  /* window of failed request */
        long pad_l3;
        long pad_l4; } XErrorEvent;
```

The serial number is the number of requests sent over the network connection since it was opened, starting from one; it is the number that was the value of **dpy→request** immediately after the failing call was made. The request code is a protocol representation of the name of the procedure that failed; these are defined in **< X/Xproto.h >**. Error codes (also defined in **< X/X.h >**) include the following:

| | | |
|---|---|---|
| **BadRequest** | 1 | Bad request code |
| **BadValue** | 2 | Integer parameter out of range |
| **BadWindow** | 3 | Parameter not a Window |
| **BadPixmap** | 4 | Parameter not a Pixmap |
| **BadBitmap** | 5 | Parameter not a Bitmap |
| **BadCursor** | 6 | Parameter not a Cursor |
| **BadFont** | 7 | Parameter not a Font |
| **BadMatch** | 8 | Parameter mismatch |
| **BadTile** | 9 | Pixmap shape invalid for tiling |
| **BadGrab** | 10 | Mouse/button already grabbed |
| **BadAccess** | 11 | Access control violation |
| **BadAlloc** | 12 | Insufficient resources |
| **BadColor** | 13 | No such color. |

**XErrorHandler** should use **XErrDescrip** to obtain textual descriptions of errors.

## XExpandEvents

**XExpandEvents ( )**

If you want to see all **MouseMoved** events, you can call **XExpandEvents** and the **MouseMoved** events are reported. You can call **XCompressEvents** to suppress all but the last **MouseMoved** event if multiple such events have been received without intervening events or replies. By default, **MouseMoved** events are compressed.

## XFeep

**XFeep** (*volume*)
int *volume*;

*volume*    Value representing the amount of sound to add to **XFeepControl**.

**XFeep** rings a bell on the keyboard.

The sound *volume* is in the range -7 to 7 (7 is the loudest) and is added to the base *volume* defined by **XFeepControl**.

## XFeepControl

**XFeepControl** (*volume*)
int *volume*;

*volume*    A value representing the amount of sound to emit for **XFeep** requests.

This function defines the base volume for **XFeep** requests. The volume is in the range 0 to 7, with 7 the loudest.

## XFetchBuffer

**char \*XFetchBuffer** (*nbytes, buffer*)
int **\****nbytes*;
int *buffer*;

*nbytes*    Returned number of bytes in the specified buffer, or 0 if the buffer is empty.

*buffer*    Buffer from which to fetch bytes.

**XFetchBuffer** retrieves the contents of the specified *buffer*. If the buffer contains data, a **malloc** is performed to get the appropriate amount of storage for the number of bytes in the **nbytes** argument. A pointer is returned to this storage, which the client must free when finished with it. If the buffer is empty, NULL is returned and *nbytes* set to 0.

Note that the buffer does not necessarily contain text, so it may contain embedded null bytes and may not terminate with a null byte.

There are eight buffers, numbered 0-7.

## XFetchBytes

**char \*XFetchBytes (*nbytes*)**
**int \****nbytes*;

*nbytes*     Returned number of bytes in buffer 0, or 0 if the buffer is empty.

**XFetchBytes** retrieves the contents of buffer 0. If the buffer contains data, a **malloc** is performed to get the appropriate amount of storage for the number of bytes in the **nbytes** argument. A pointer is returned to this storage, which the client must free when finished with it. If the buffer is empty, NULL is returned and *nbytes* set to 0.

Note that the buffer does not necessarily contain text, so it may contain embedded null bytes and may not terminate with a null byte.

## XFetchName

**Status XFetchName (*w, name*)**
**Window *w*;**
**char \*\****name*;

*w*          Window to be null-terminated.

*name*    Address of a pointer to the name of a window.

**XFetchName** sets *name* to a pointer to the name of the window (a null-terminated string). If no name was ever set, it sets *name* to NULL. The client must free the name string when finished with it.

**XFetchName** returns 0 if it fails, non-zero otherwise. If the window has never had a name set, this is not considered a failure, and **XFetchName** returns a non-zero status.)

## XFlush

**XFlush ( )**

**XFlush** sends all output requests that have been buffered but not yet sent. Flushing is done automatically the next time input is read (with **XPending**, **XNextEvent**, **XPeekEvent**, **XCheckMaskEvent**, **XMaskEvent**, **XCheckWindowEvent**, or **XWindowEvent**), so most clients should not need to use this function.

This function is sometimes helpful in debugging programs, but causes a significant degradation in performance.

## XFocusKeyboard

**XFocusKeyboard (*w*)**
**Window *w*;**

*w*          Window to designate as the ***input focus*** window.

**XFocusKeyboard** designates a window as the input focus window. If the window that would normally receive a **KeyPressed** or **KeyReleased** event is not the focus window or one of its descendents, the event is sent to the focus window instead. The events go to the client that has selected input on the focus window. In general, this may be a client other than the one that has called **XFocusKeyboard**. For instance, a window manager may allow the user to designate an arbitrary window as the keyboard focus. The root window is the default focus window. If the focus window is closed, the closest existing ancestor inherits the input focus.

## XFontWidths

**short \*XFontWidths (*font*)**
**Font *font*;**

*font*      Font ID for which character widths are requested.

**XFontWidths** allocates memory for and returns a pointer to an array containing the width of every character defined in the font. This function can be used only for variable-width fonts. If **XFontWidths** returns NULL, an error has occurred and no array is allocated. The client must free this array when it is no longer needed.

**XFontWidths** should be used in conjunction with **XQueryFont**, which returns (among other data) the font's *firstchar* and *lastchar*. The length of the array returned by

**XFontWidths** is always equal to *lastchar - firstchar + 1.* In the array, widths[i] is set to the width of character [firstchar + i].

# XFreeBitmap

**XFreeBitmap** (*bitmap*)
**Bitmap** *bitmap*;

*bitmap*    Identifies the bitmap for which storage is to be freed.

**XFreeBitmap** frees all the storage associated with the specified Bitmap. The Bitmap should never be referenced again.

# XFreeColors

**XFreeColors** (*pixels, npixels, planes*)
**int** *pixels*[ ];
**int** *npixels*;
**int** *planes*;

*pixels*    Pointer to pixel values whose color map cells are to be freed.

*npixels*   Number of colors.

*planes*    Plane mask.

This function frees color map cells. The cells represented by pixels whose values are in the array are freed. If any planes are specified, they are freed.

# XFreeCursor

**XFreeCursor** (*cursor*)
**Cursor** *cursor*;

*cursor*    Cursor to free.

The specified cursor is destroyed, and should not be referred to again or an error is generated.

## XFreeFont

**XFreeFont** (*font*)
**Font** *font*;

*font*  Font ID to be freed.

**XFreeFont** tells the server that this font is no longer needed. The font may be unloaded on the server if this is the last reference to the font. In any case, the font should never again be referenced.

## XFreePixmap

**XFreePixmap** (*pixmap*)
**Pixmap** *pixmap*;

*pixmap*  Pixmap for which storage is to be freed.

**XFreePixmap** frees all the storage associated with this Pixmap. The Pixmap should never be referenced again.

## XGeometry

**XGeometry** (*position, default, bwidth, fwidth, fheight, xadd, yadd, x, y, width, height*)
**char** \**position*, \**default*;
**int** *bwidth*;
**int** *fwidth, fheight*;
**int** *xadd, yadd*;
**int** \**x*, \**y*, \**width*, \**height*;

*position* Incompletely specified geometry.

*default*  Fully qualified default geometry.

*bwidth*  Width of the window border.

*fwidth*  Unit size of the width specified in geometry.

*fheight* Unit size of the height specified in geometry.

*xadd*  Additional space (in pixels) to add to the interior width.

*yadd*  Additional space (in pixels) to add to the interior height.

| $x$ | Returned value indicating the x coordinate of the window. |
|---|---|
| $y$ | Returned value indicating the y coordinate of the window. |
| *width* | Returned value indicating the inside width of the window. |
| *height* | Returned value indicating the inside height of the window. |

This routine does all the work required to determine the placement of a window using the current format to position windows. Given a fully qualified default geometry specification and an incompletely specified geometry specification, this function returns a bitmask value as defined in the **XParseGeometry** call. User programs typically use the **XCreate** or **XCreateTerm** functions to create the window.

If the function returns either the **XValue** or **YValue** flag, you should place the window at the requested position. The border width, additional interior space, and unit sizes are passed in to facilitate computation of the window size.

Unit sizes are usually font size for text windows and pixel size for graphics windows. The *xadd* and *yadd* parameters are typically used when the unit sizes are not 1.

See "Geometry Specification" on page 3-4 for more details on geometry.

# XGetColor

**Status XGetColor** (*colorname, harddef, exactdef*)
**char** *\*colorname*;
**Color** *\*harddef*;
**Color** *\*exactdef*;

*colorname* Color name to search for.

*harddef* Returned color definition indicating the color closest in appearance to *colorname* supported by this particular hardware.

*exactdef* Returned exact color definition from the database definition for the supplied *colorname* argument.

Applications often need to know what the correct value of **red** may be on a particular display in order to provide a good user interface. Given a text string (*colorname*, for example **red**), this function returns the Color structure in the supplied structure. It uses a database on the server to resolve the color by name from the file **/usr/lpp/X/rgb/rgb**. A text representation of this file can be found in **/usr/lpp/X/rgb/rgb.txt**. This function returns 0 if unsuccessful, or non-zero if successful.

Both the exact data base definition and the closest color supported by the hardware are returned.

## XGetColorCells

> **Status XGetColorCells** (*contig, ncolors, nplanes, planes, pixels*)
> int *contig*;
> int *ncolors*;
> int *nplanes*;
> int *\*planes*;
> int *pixels[ncolors]*;
>
> *contig*   Set to 1 if planes must be contiguous, 0 if not contiguous.
>
> *ncolors*   Number of colors.
>
> *nplanes*   Number of planes.
>
> *planes*   Plane mask.
>
> *pixels*   Pointer to the returned pixel values.
>
> This function allocates **n\*2$^P$** color map cells, where **n** (**ncolors**) is the number of colors and P (**nplanes**) is the number of planes specified. This function returns a plane mask, which can be contiguous if requested. Additional pixel values are obtained by ORing one or more bits from the plane mask. The initial colors for all of these cells is undefined.
>
> If zero colors are requested, the request allocates all cells with a pixel value having at least one non-zero bit in the plane mask. At most one such request succeeds. This is typically your favorite window manager. Allocations are automatically deallocated when clients exit.

## XGetDefault

> **char XGetDefault** (*command, keyword*)
> char *\*command*;
> char *\*keyword*;
>
> *command*  Pointer to the command name.
>
> *keyword*   Pointer to the keyword name.
>
> The number of options that a program may need can be very large in the X environment. These options include fonts of various sorts, colors of characters, mouse function, background, text, cursor, and so on.
>
> **XGetDefault** makes it easier to find out what the user wants as the default font, colors, and so on.

**XGetDefault** returns NULL if no option of the specified keyword exists for the command. Defaults are read out of a file called **.Xdefaults** in the user's home directory. See "Creating the Default File" on page 2-4 for details of its format.

The strings returned by **XGetDefault** are owned by **Xlib** and should not be modified or freed by the client.

# XGetFont

Font XGetFont (*name*)
char *_name_;

*name*      Name of the requested font.

**XGetFont** loads a font of the specified **name**. A font ID is returned if successful, or 0 if unsuccessful. The client should call **XFreeFont** when the font is no longer needed.

# XGetHardwareColor

Status XGetHardwareColor (*def*)
Color *_def_;

*def*      Pointer to the color definition.

When passed a color definition structure **def**, with red, green and blue values set, this function fills in the pixel value with the closest color provided by the hardware. The corresponding color map cell is read-only. The function returns 0 if unsuccessful (typically due to lack of resources) or non-zero if successful.

Read-only color map cells are shared among clients. When the last client deallocates a shared cell, it is deallocated.

# XGetHosts

#include < sys/socket.h >

struct in_addr *XGetHosts (*nhosts*)
int *_nhosts_;

*nhosts*      Returned number of hosts that can open connections.

# XGetResizeHint

XGetHosts returns the current list of hosts allowed to make connections. Memory is allocated for and a pointer is returned to an array that contains each host in the list. The *nhosts* value indicates the number of elements in the array. The memory used by this function should be freed when no longer needed.

## XGetResizeHint

**XGetResizeHint** (*w, width0, height0, widthinc, heightinc*)
**Window** *w*;
**int** \**widthc*, \**heightc*;
**int** \**widthinc* \**heightinc*;

*window*    Window being queried.

*widthc, heightc* Returned base size of window.

*widthinc, heightinc* Returned increment values used to compute resize.

**XGetResizeHint** asks for the window's resize parameters and assigns them to the client's variables.

## XGrabButton

**Status XGrabButton** (*w, cursor, buttonMask, eventMask*)
**Window** *w*;
**Cursor** *cursor*;
**int** *buttonMask*;
**int** *eventMask*;

*w*        Window to send grabbed events to.

*cursor*    Cursor to use if the grab is successful.

*buttonMask* Button mask bits.

*eventMask* Mask that determines which mouse events are reported after the mouse is grabbed.

After **XGrabButton** is called, the mouse is automatically grabbed whenever a particular mouse button is pressed while certain keys are down. The combination is specified in **buttonMask**. This mask must have exactly one of the **LeftMask**, **MiddleMask**, and **RightMask** bits set, and may have some combination of the **ControlMask**, **MetaMask**, **ShiftLockMask**, **ShiftMask**, and **AltGraphMask** bits set as well.

If the specified button is pressed while exactly the specified keys are down, this and all future mouse events are grabbed until all buttons are released, with events sent to windows as described under **XGrabMouse**.

The event mask determines the mouse events reported while the mouse is grabbed.

An error occurs if another client has already grabbed the same button/key combination and has not ungrabbed it.

Note that this procedure returns a status and is therefore synchronous. The function returns 0 if the button could not be grabbed, non-zero if the the button was grabbed successfully.

## XGrabMouse

**Status XGrabMouse** (*w, cursor, mask*)
**Window** *w*;
**Cursor** *cursor*;
**int** *mask*;

*w*       Window to send grabbed events to.

*cursor*  Cursor to use if grab is successful.

*mask*    Event mask.

After **XGrabMouse** is called, all future mouse events go only to windows for which the client has previously called **XSelectInput**. The **ButtonPressed**, **ButtonReleased**, **EnterWindow**, **LeaveWindow**, **MouseMoved**, **LeftDownMotion**, **MiddleDownMotion**, and **RightDownMotion** bits of the *mask* parameter temporarily override the corresponding bits in any mask previously passed to **XSelectInput**; other bits in *mask* are ignored.

If one of the aforementioned events occurs, and the client has not called **XSelectInput** on the window where the event would normally be sent, then the event is sent to the window *w*, provided that the event is specified in *mask* and is not **EnterWindow** or **LeaveWindow**.

An error occurs if a different client has already grabbed the mouse and has not ungrabbed it. No error results from the same client grabbing the mouse more than once without ungrabbing it in between. A mouse-grabbing client may want to do this in order to change the cursor or event mask without ungrabbing the mouse.

Grabbing the mouse overrides any **XGrabButton** calls previously issued by this or any other client, until the mouse is ungrabbed.

Note that this procedure returns a status and is therefore synchronous, even though no other values are returned. The function returns 0 if the mouse could not be grabbed, non-zero if the mouse was successfully grabbed.

## XGrabServer

**XGrabServer ( )**

This request can be used to control processing of output on other connections by the X Server. No processing of requests or close downs on all other connections occurs while the server is grabbed.

This may be useful for menus or window manager programs who may want to preserve bits on the screen while temporarily suspending processing on other connections.

## XInterpretLocator

**Status XInterpretLocator (w, x, y, subw, loc)**
**Window** *w*;
**int \*x, \*y;**
**Window \*subw;**
**Locator** *loc*;

*w*      Window for which locator is interpreted.

*x, y*     Returned coordinates relative to the upper left-hand corner of *w*.

*subw*    Identifies the child window if absolute coordinates fall within it; otherwise, this returned value is set to 0.

*loc*      Locator ( (x < < 16) | y).

**XInterpretLocator** converts absolute coordinates to coordinates relative to the top left inner corner of a window. These window-relative coordinates are assigned to the variables *x* and *y*. Normally, the Locator will be obtained from an input event rather than constructed from a separate *x* and *y* coordinate. It is seldom useful to deal with absolute coordinates, but if you must convert absolute coordinates to a locator, a locator is constructed from a coordinate pair by (x< <16) | y. If the absolute coordinates also fall within a child window, the routine sets *subw* to that child window; otherwise it sets *subw* to 0.

## XIOErrorHandler

**XIOErrorHandler** (*handler*)
int *handler(Display \*)*;

*handler*     Error handler supplied by the program.

The program's supplied error handler is called by **Xlib** if any sort of system call error (such as losing the connection to the server) occurs. A system call error is assumed to be a fatal condition and the called I/O error handler should not return. If the I/O error handler does return, the client process will exit.

## XKeyClickVolume

**XKeyClickVolume** (*volume*)
int *volume*;

*volume*     A value in the range 0 (no volume) to 8 (loudest volume) that controls the volume of the click resulting from a pressed key.

**XKeyClickVolume** controls the volume of the click that the keyboard makes when a key is pressed.

## XLine

**XLine** (*w, x1, y1, x2, y2, width, height, pixel, func, planes*)
**Window** *w*;
int *x1, y1, x2, y2*;
int *width, height*;
int *pixel*;
int *func*;
int *planes*;

*w*           Window in which to draw a line.

*x1, y1, x2, y2*
               Identifies the beginning pixel (x1 and y1) and the end pixel (x2 and y2) between which a line is drawn.

*width, height*
               Specifies the height and width of the brush to be drawn along the line.

# XLine

pixel     Source pixel value (color).

func      Display function.

planes    Plane mask.

**XLine** draws a line from pixel (*x1, y1*) to pixel (*x2,y2*) inclusive in the specified window, using the specified display function *func*.

The line will be drawn in the color specified by the *pixel* value. The *width* and *height* arguments specify the size of a brush to be drawn along the line.

# XLockToggle

**XLockToggle ( )**

This function changes the mode of the **Caps Lock** key on the keyboard. In **LockToggle** mode, **KeyPressed** and **KeyReleased** events are never sent for the **Caps Lock** key, and the state of the **ShiftLockMask** sent in events is toggled on every press of the **Caps Lock** key.

# XLockUpDown

**XLockUpDown ( )**

This function changes the mode of the **Caps Lock** key on the keyboard. When the keyboard is in **LockUpDown** mode, **KeyPressed** and **KeyReleased** events are sent as for any other key, and the **ShiftLockMask** sent in events gives the current state of the key.

# XLookUpAssoc

**caddr_t XLookUpAssoc** (*table, xid*)
**XAssocTable *****table**;
**XId** *xid*;

table     Pointer to an association table.

xid       X resource ID.

This function retrieves the data stored in an **XAssocTable** by its XId. If a matching XId can be found in the table, the routine returns a pointer to the data associated with it. If the XId cannot be found in the table, the routine returns NULL.

# XLookUpMapping

char **XLookUpMapping** (*event, nbytes*)
**XKeyPressedEvent** *\*event*;
int *\*nbytes*;

*event*    Pointer to the character string mapped to this event.

*nbytes*    Pointer to the number of bytes in the character string, or 0 if no text is mapped to the event.

This function maps events to counted character strings (an array of characters and the length; the null character is legitimate in this use). The function returns a pointer to a static counted character string which must not be modified by a client, and the number of bytes in the string.

This function searches for the current keyboard mapping in the following order:

- **$XDIR/.Xkeymap**
- **$HOME/.Xkeymap**
- **/usr/lpp/Xdefaults/.Xkeymap**

If these files are not present, **XLookupMapping** defaults to a built-in table. If no text is defined for a particular key, *nbytes* is zero.

The **.Xkeymap** file is produced by the **keycomp** program, which reads a text file of keyboard mappings. The directory **/usr/lpp/X/defaults** contains the keyboard mappings for all supported languages. The function performs normal interpretation of shift bits (meta, shift, shift lock, control). **XLookupMapping** supports **Alt-NumPad** and **NumLock** key processing, as well as dead key processing as defined in **keycomp**.

**Alt-NumPad** processing begins when the first **Alt-NumPad** key is pressed and ends when either the third **Alt-NumPad** key is pressed or a non-**Alt-NumPad** key is pressed. The final keycode is not returned to the user until one of these terminating events occurs. If the terminating event is a non-**Alt-NumPad** key, then both the generated **Alt-NumPad** key code and the string of the non-**Alt-NumPad** key is returned in a single buffer.

In order for this processing to work correctly, both the **Alt** key and the **NumPad** key (in **Alt** state) must be defined as UNBOUND in the source keymap (see **keycomp(1)**). In addition, this function tracks the **NumLock** state only if the **NumLock** key is defined as UNBOUND.

The user should do a **strncpy** to copy the result (if needed) to his own storage if the data must be modified. If a different keymap file is desired, it can be set using **XUseKeymap**.

## XLowerWindow

**XLowerWindow** (*w*)
**Window** *w*;

*w*          Window to be lowered.

**XLowerWindow** lowers this window so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack, while leaving its x and y location on the desk constant.

Lowering a mapped opaque window generates exposure events on any opaque windows it formerly obscured.

Lowering a transparent window does not affect the screen.

## XMakeAssoc

**XMakeAssoc** (*table, xid, data*)
**XAssocTable** *\*table*;
**XId** *xid*;
**caddr_t** *data*;

*table*       Pointer to an association table.

*xid*         X resource ID.

*data*        Pointer to the data to associate with the XId.

This function inserts data into an **XAssocTable** keyed on an XId. Data is inserted into the table only once. Redundant inserts are meaningless. The queue in each association buffer is sorted from the lowest XId to the highest XId.

## XMakePattern

**Pattern XMakePattern** (*pattern, length, multiplier*)
**int** *pattern, length, multiplier*;

*pattern*     Bit string.

*length*      Length of bit string (16 bits maximum).

*multiplier*  Specifies the number of times a bit is repeated before moving to the next bit.

In this function, *pattern* is a bit string of the specified *length* (at most 16 bits). The *multiplier* specifies how many times each bit in the string is repeated (maximum of 4096 times) before moving to the next bit. The least-significant bits are processed first, and repeated as many times as needed.

## XMakePixmap

**Pixmap XMakePixmap** (*bitmap, fg, bg*)
**Bitmap** *bitmap*;
int *fg, bg*;

*bitmap*    Binary mapping of pixel values.

*fg*    Foreground pixel value.

*bg*    Background pixel value.

**XMakePixmap** returns a Pixmap constructed from a bitmap and two pixel values. A 1 in the bitmap means the pixmap will have a pixel value of the **foreground**; a 0 in the bitmap means the pixmap will have a pixel value of the **background**. If 0 is specified for the **bitmap** argument, it returns a Pixmap of indefinite size suitable for use as a constant tiling pixmap.

If the Bitmap is the same size as a tile, then the Pixmap can be used as a tile.

## XMakeTile

**Pixmap XMakeTile** (*pixel*)
int *pixel*;

*pixel*    Pixel value to use for the Pixmap.

This function returns a constant color Pixmap suitable for use as a tiling argument.

## XMapSubwindows

> **XMapSubwindows** (*w*)
> **Window** *w*;

*w*        Window for which subwindows are to be mapped.

**XMapSubwindows** maps all subwindows of the specified window in an unpredictable order.

This function also generates an **ExposeWindow** event on each newly-displayed opaque window.

Note that this is more efficient than mapping many windows one at a time, as much of the work need only be performed once for all of the windows rather than for each window.

## XMapWindow

> **XMapWindow** (*w*)
> **Window** *w*;

*w*        Window to be mapped.

**XMapWindow** maps the window and raises the window and all of its subwindows which have had map requests to the top of the stack of windows. A subwindow will appear on the screen so long as all of its ancestors are mapped. The previous contents of all opaque windows are lost; mapping transparent windows does not affect the screen.

Mapping a window that has an unmapped ancestor does not display the window, but marks it as eligible for display when the ancestor becomes mapped.

Mapping an already mapped window has no effect (the window is not raised).

If the window is opaque, **XMapWindow** generates **ExposeWindow** events on each opaque window that it displays. If the client first maps the window, then paints the window, then begins processing input events, the window will be painted twice. To avoid this situation, the client should do one of the following:

- First Map, then call **XSelectInput** for exposure events, then repaint the windows explicitly.
- First call **XSelectInput** for exposure events, then map, then process input events normally.

The event list will include **ExposeWindow** events for each window that has appeared on the screen; the client's normal response to an **ExposeWindow** event should be to repaint the window. The second method is preferred, as it usually leads to simpler programs.

# XMaskEvent

**XMaskEvent** (*mask, rep*)
int *mask*;
**XEvent** *\*rep*;

*mask*     Mask to identify the event for which to look.

*rep*      Pointer to an **XEvent**.

This subroutine is used to look for specific events. **XMaskEvent** flushes the output buffer, then removes the next event in the queue which matches the passed mask. The event is copied into an **XEvent** supplied by the caller. Events earlier in the queue are not discarded. If no such event has been queued, **XMaskEvent** blocks until one is received.

# XMouseControl

**XMouseControl** (*acceleration, threshold*)
int *acceleration, threshold*;

*acceleration* Specifies the rate of cursor movement as related to mouse movement.

*threshold* Number of pixels required before mouse is moved.

This function defines how the mouse moves. The *acceleration* is a multiplier for movement. For example, specifying **3** means the cursor moves three times as fast as the mouse. Acceleration only takes effect if the mouse moves more than *threshold* pixels at once, and only applies to the amount beyond the *threshold*.

# XMoveArea

**XMoveArea** (*w, sx, sy, dx, dy, width, height*)
**Window** *w*;
int *sx, sy, dx, dy, width, height*;

*w*        Window in which the move is to be made.

*sx, sy*    Coordinates of the area to be moved.

*dx, dy*    Coordinates of the location to which to move the specified area.

*width, height* Width and height of the area to be moved.

# XMoveArea

**XMoveArea** moves one region of the window to another (possibly overlapping) region of the same window, using the supplied display function *func*.

If parts of the source region are obscured, the corresponding parts of the destination are painted with the background tile. If a client has called **XSelectInput** on this window with the **ExposeCopy** bit set, then **ExposeRegion** events will be generated on any such parts of the destination, and then an **ExposeCopy** event will be generated. All of these events are guaranteed to be together in the stream, with no intervening events. This sequence makes it possible to *scroll* the contents of a window, getting exposure events from wherever the window was obscured to refresh those areas of the screen.

# XMoveWindow

**XMoveWindow** (*w, x, y*)
**Window** *w*;
**int** *x, y*;

*w*        Window to be moved.

*x, y*      Top-left coordinate of the new location of the window.

**XMoveWindow** moves and raises the window, without changing its size. This function does not change the mapping state of the window. The *x* and *y* coordinates are the new location of the top-left pixel of the window's border (or the window itself, if it has no border). Moving a mapped-opaque window may lose its contents if the window's tile mode is relative or if the window is obscured by non-children. Moving a transparent window does not affect the screen. If the contents are lost, exposure events will be generated for the window and any mapped opaque subwindows.

Moving a mapped-opaque window will generate exposure events on any formerly obscured opaque windows.

# XNextEvent

**XNextEvent** (*rep*)
**XEvent** *\*rep*;

*rep*      Pointer to an **XEvent**.

**XNextEvent** flushes the output buffer, then removes an input event from the head of the queue and copies it into an **XEvent** supplied by the caller. If the queue is empty, **XNextEvent** blocks until an event is received.

# XOpenDisplay

**Display \*XOpenDisplay** (*name*)
**char \****name***;

*name*      Name of the server to which a connection is to be opened.

This function takes the *name* of the server and opens a connection to the server for the display hardware. If the display name string is NULL, it uses the environment variable DISPLAY to determine the display and communications domain to use. The display string or DISPLAY environment variable should be in the format hostname:number, where hostname is the name of a machine and number is the number of the display on that machine. For example, site2a:3 would be display 3 on the machine **site2a**. **XOpenDisplay** connects through TCP or local streams to the server. If *hostname* is unix, local domain IPC is used.

If the call is successful, it returns a pointer to a Display structure, which is defined in <X/Xlib.h>. The procedure returns NULL on failure.

Macros defined in "Macros and Constants" on page 4-30 allow access to the Display structure.

Other elements of the **Display** structure are private to the X library and must not be used.

# XOpenFont

**FontInfo XOpenFont** (*name*)
**char \****name***;

*name*      Pointer to the location of the font storage.

This function does a **XGetFont**, **XQueryFont** and **XFontWidths** in one operation, creating an instance of the font structure. The function allocates the memory in which to store the font information, and returns NULL if unsuccessful.

## XParseColor

>Status **XParseColor** (*spec, def*)
>char \**spec*;
>Color \**def*;

>*spec*    Character string specifying the color.

>*def*    Returned color definition.

>This subroutine provides a user interface to select colors. The function takes a string specification of a color, typically from a command line or **XGetDefault** option, and returns the corresponding red, green, and blue values that are suitable for a subsequent call to **XGetHardwareColor** or **XStoreColor**.  The color can be specified either as a color name (as in **XGetColor**), or as an initial sharp sign character following by a numeric specification in one of the following formats:

>| | |
>|---|---|
>| #RGB | 4 bits each |
>| #RRGGBB | 8 bits each |
>| #RRRGGGBBB | 12 bits each |
>| #RRRRGGGGBBBB | 16 bits each |

>In the table above, R, G, and B represent single hexadecimal digits (upper- or lowercase). When fewer than 16 bits each are specified, they represent the most significant bits of the value.  For example, #3a7 is the same as #3000a0007000.

>This routine fails if the initial character is a sharp sign but the string otherwise fails to fit of the above formats, or if the initial character is not a sharp sign and the named color does not exist in the server's database.

## XParseGeometry

>int **XParseGeometry** (*string, x, y, width, height*)
>char \**string*;
>int \**x*, \**y*,\**width*, \**height*;

>*string*    Geometry specification.

>*x, y, width, height* Returned coordinates and size of the window.

>By convention, X Windows applications use a standard string to indicate window size and placement.  This subroutine facilitates conformance to this standard.  It is not normally used by user programs, which typically use the **XCreate** or **XCreateTerm** subroutines to create the window.  This subroutine is used to parse strings of form
>=<width>x<height>{+-}<xoffset>{+-}<yoffset>, where width, height, xoffset and yoffset are returned in the *width*, *height*, *x* and *y* arguments.  It returns a bitmask that

indicates which of the four values were actually found in the string, and whether the x and y values are negative (remember, -0 is not equal to +0 in this system). For each value found, the corresponding argument is updated; for each value not found, the argument is left unchanged. The bits are **XValue**, **YValue**, **WidthValue**, **HeightValue**, **XNegative**, **YNegative**, and are defined in **<X/Xlib.h>**. They will be set whenever one of the values is defined or signs are set.

For more information on geometry, see "Geometry Specification" on page 3-4.

## XPeekEvent

**XPeekEvent** (*rep*)
**XEvent** *\*rep*;

*rep*          Pointer to an **XEvent**.

**XPeekEvent** flushes the output buffer, then copies the input event from the head of the queue into an **XEvent** supplied by the caller without removing the input event from the queue. If the queue is empty, **XPeekEvent** blocks until an input event is received. To determine if a queue has any input events to peek, use the **QLength( )** macro.

## XPending

**XPending ( )**

**XPending** flushes the output buffer, then returns the number of input events that have been received from the server, but not yet removed from the queue. (Events are removed from the queue by calling **XNextEvent** or **XWindowEvent**.)

You should always call **XPending** before calling **select** on the file descriptor contained in the display structure. The input you are trying to wait for may have already arrived and be sitting in Xlib's queue. Another strategy might be to call **XFlush** after finding out if there are any unprocessed events on the queue by using the **QLength** macro before calling **select**.

## XPixFill

XPixFill (*w, x, y, width, height, pixel, clipmask, func, planes*)
**Window** *w*;
**int** *x, y, width, height*;
**int** *pixel*;
**Bitmap** *clipmask*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Destination window. |
| *x, y* | Upper-left coordinates of the region to fill. |
| *width* | Width of the region to be filled. |
| *height* | Height of the region to be filled. |
| *pixel* | Source value to use to fill the region. |
| *clipmap* | Clip mask. |
| *func* | Display function. |
| *planes* | Plane mask. |

This function performs a display function in a region of the window. The *pixel* value is used as the source. If a *clipmask* bitmap is specified, it defines the shape of the source and which pixels of the destination will be affected. This can be useful for defining non-rectangular icons.

## XPixSet

XPixSet (*w, x, y, width, height, pixel*)
**Window** *w*;
**int** *x, y, width, height*;
**int** *pixel*;

| | |
|---|---|
| *w* | Destination window. |
| *x, y* | Upper-left coordinates of the region to fill. |
| *width* | Width of the region to be filled. |
| *height* | Height of the region to be filled. |
| *pixel* | Source value to use to fill the region. |

XPixSet is equivalent to **XPixFill** with *func* = **GXcopy**, *planes* = **AllPlanes** and *clipmask* = **NULL**.

## XPixmapBitsPutXY

**XPixmapBitsPutXY** (*w, x, y, width, height, data, clipmask, func, planes*)
**Window** *w*;
**int** *x, y, width, height*;
**short** *\*data*;
**Bitmap** *clipmask*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Destination window. |
| *x, y* | Upper-left coordinates of the region at which to place Pixmap. |
| *width* | Width of the region to be filled. |
| *height* | Height of the region to be filled. |
| *data* | Pointer to the source Pixmap. |
| *clipmask* | Clip mask. |
| *func* | Display function. |
| *planes* | Planes mask. |

**XPixmapBitsPut** copies a client-supplied XY format Pixmap into a window according to the specified display function.

See "Pixmaps" on page 4-16 for the definition of XY format pixmaps.

The area modified is controlled by the *clipmask* argument, if it is not zero. Only the bits in the *clipmask* are modified in the window. This is often useful for icon generation.

## XPixmapBitsPutZ

**XPixmapBitsPutZ** (*w, x, y, width, height, data, clipmask, func, planes*)
**Window** *w*;
**int** *x, y, width, height*;
**short** *\*data*;
**Bitmap** *clipmask*;
**int** *func*;
**int** *planes*;

# XPixmapBitsPutZ

| | |
|---|---|
| *w* | Destination window. |
| *x, y* | Upper-left coordinates of the region at which to place Pixmap. |
| *width* | Width of the region to be filled. |
| *height* | Height of the region to be filled. |
| *data* | Pointer to the source Pixmap. |
| *clipmask* | Clip mask. |
| *func* | Display function. |
| *planes* | Plane mask. |

**XPixmapBitsPut** copies a client-supplied Z format Pixmap into a window according to the specified display function.

See "Pixmaps" on page 4-16 for the definition of a Z format pixmap.

The area modified is controlled by the *clipmask* argument, if it is nonzero. Only the bits in the *clipmask* are modified in the window. This is often useful for icon generation.

# XPixmapGetXY

**XPixmapGetXY** (*w, x, y, width, height, data*)
**Window** *w*;
**int** *x, y, width, height*;
**short** *\*data*;

| | |
|---|---|
| *w* | Source window. |
| *x, y* | Upper-left coordinates of the region to get. |
| *width* | Width of the region to get. |
| *height* | Height of the region to get. |
| *data* | Pointer to the destination pixmap. |

**XPixmapGet** returns the pixmap in the XY format into the specified area of memory.

See "Pixmaps" on page 4-16 for information on determining how much memory to reserve for the returned data. This function is intended for window dump purposes.

The window must be mapped. If the window has no subwindows or overlapping windows, the specified portion of the window must be fully visible on the screen.

## XPixmapGetZ

**XPixmapGetZ** (*w, x, y, width, height, data*)
**Window** *w*;
**int** *x, y, width, height*;
**short** *\*data*;

| | |
|---|---|
| *w* | Source window. |
| *x, y* | Upper-left coordinates of the region to get. |
| *width* | Width of the region to get. |
| *height* | Height of the region to get. |
| *data* | Pointer to the destination pixmap. |

**XPixmapGet** returns the pixmap in the Z format into the specified area of memory.

See "Pixmaps" on page 4-16 for information on determining how much memory to reserve for the returned data. This function is intended for window dump purposes.

The window must be mapped. If the window has no subwindows or overlapping windows, the specified portion of the window must be fully visible on the screen.

## XPixmapPut

**XPixmapPut** (*w, sx, sy, dx, dy, width, height, pixmap, func, planes*)
**Window** *w*;
**int** *sx, sy*;
**int** *dx, dy*;
**int** *width, height*;
**Pixmap** *pixmap*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Destination window. |
| *sx, sy* | Coordinates of a region within a Pixmap. |
| *dx, dy* | Coordinates of a region within a window. |
| *width, height* | Size of the region. |
| *pixmap* | Pixmap to be put. |
| *func* | Display function. |
| *planes* | Plane mask. |

**XPixmapPut** performs a display function on a specified region of the pixmap and a specified region of the window.

## XPixmapSave

**Pixmap XPixmapSave** (*w, x, y, width, height*)
**Window** *w*;
**int** *x, y, width, height*;

*w*          Source window.

*x, y*       Upper left coordinates of the region to be saved.

*width, height* Width and height of the region to save.

**XPixmapSave** creates a Pixmap from the given portion of the window. The pixmap will contain a direct image of that portion of the screen, including any visible portions of subwindows or overlapping windows, so this routine should be used with caution. Its main use will probably be in conjunction with **XUnmapTransparent**, or in implementing pop-up menus or other temporary windows that save the bits under them and then restore those bits when destroyed.

The window must be mapped. If the window has no overlapping windows or subwindows, the specified portion of the window would be fully visible on the screen.

This function returns the Pixmap ID for the saved pixmap if successful, or 0 if unsuccessful.

## XPutBackEvent

**XPutBackEvent** (*event*)
**XEvent** *\*event*;

*event*      Pointer to an **XEvent**.

**XPutBackEvent** pushes an event back onto the head of the current display's input queue. This can be useful to recall events at a later time after reading them.

## XQueryBrushShape

**XQueryBrushShape** (*width, height, rwidth, rheight*)
int *width, height*;
int * *rwidth*, \**rheight*;

*width, height* Requested brush size.

*rwidth, rheight* Brush size supported by the hardware that is closest to *width, height*.

This function returns the closest shape actually supported by the display hardware for brushes, since not all hardware is capable of supporting arbitrary size brushes. Painting lines using brushes of widths not supported by the hardware has unpredictable results.

## XQueryColor

**Status XQueryColor** (*def*)
**Color** \**def*;

*def*        Returned color definition for a pixel value.

This function returns the color values for the pixel value specified in *def*. The function returns the definition if successful, or 0 if unsuccessful.

## XQueryColors

**XQueryColors** (*defs, ncolors*)
**Color** *defs[ ]*;
int *ncolors*

*defs*        Returned color definitions for a pixel value.

*ncolors*    Number of color definitions.

This function returns the color values for the pixel values specified in *defs*[ ].

## XQueryCursorShape

**XQueryCursorShape** (*width, height, rwidth, rheight*)
**int** *width, height*;
**int** *\*rwidth, \*rheight*;

*width, height* Requested cursor size.

*rwidth, rheight* Cursor size supported that is closest to *width, height*.

This call provides a way to determine the cursor shapes supported by the display. Some displays allow larger cursors than other displays. This function returns the closest shape cursor actually supported by the display hardware. This function returns a size acceptable for **XStoreCursor**. Applications should be prepared to use smaller cursors on displays that cannot support large ones.

## XQueryFont

**Status XQueryFont** (*font, info*)
**Font** *font*;
**FontInfo** *\*info*;

*font*     Font to query for information.

*info*     Returned pointer to the font information structure. See "Font" on page 4-27 for a definition of the structure.

**XQueryFont** gets various facts about a font and returns the information in the client-passed structure **FontInfo**. This function does not get the font width array.

## XQueryInput

**XQueryInput** (*w, mask*)
**Window** *w*;
**int** *\*mask*;

*w*        Window whose mask is queried.

*mask*     Returned input event mask for the specified window.

**XQueryInput** returns the event mask currently in effect for the specified window. The bits of the mask are defined in "Events" on page 4-18.

## XQueryMouse

**Status XQueryMouse** (*w, x, y, subw*)
**Window** *w*;
**int** \**x* \**y*;
**Window** \**subw*;

*w*　　　　Mouse position is calculated relative to this window.

*x, y*　　　Returned coordinates of the mouse relative to the top-left inside corner of *w*.

*subw*　　Returned ID of a child window, if the mouse is also in a child window.

**XQueryMouse** determines the current mouse coordinates. The coordinates returned are relative to the top-left inside corner of the window, even if the mouse is outside the window (coordinates can be negative). If the mouse is also in a child window, then *subw* is set to that child, otherwise *subw* is set to 0.

## XQueryMouseButtons

**Status XQueryMouseButtons** (*w, x, y, subw, state*)
**Window** *w*;
**int** \**x* \**y*;
**Window** \**subw*;
**short** \**state*;

*w*　　　　Mouse position is calculated relative to this window.

*x, y*　　　Returned coordinates of the mouse relative to the top-left inside corner of *w*.

*subw*　　Returned ID of a child window, if the mouse is also in a child

*state*　　Returned mask indicating which mouse buttons are pressed.

This function returns the same information as **XQueryMouse**, with the addition of the state of the mouse buttons. The mask format is described in "Events" on page 4-18.

## XQueryTileShape

**XQueryTileShape** (*width. height, rwidth, rheight*)
int *width, height*;
int * *rwidth, *rheight*;

*width, height* Requested tile size.

*rwidth, rheight* Closest supported tile size.

This function sets *rwidth* and *rheight* to the tile size closest to the requested tile size. Not all hardware will allow pixmaps of arbitrary shapes for tiling.

## XQueryTree

**Status XQueryTree** (*w, parent, nchildren, children*)
**Window** *w*;
**Window** *\*parent*;
int *\*nchildren*;
**Window** *\*\*children*;

*w*          Window to query.

*parent*     Returned ID of the parent of the window.

*nchildren*  Returned number of children for the window.

*children*   Returned pointer to a list of children, if any.

**XQueryTree** returns a list of children of the specified window, its parent, and the number of children of this window. The function returns a pointer to a list of the children windows. The list must be deallocated when no longer needed.

## XQueryWidth

int **XQueryWidth** (*str, font*)
char *\*str*;
**Font** *font*;

*str*        Pointer to a null-terminated string.

*font*       Font ID.

**XQueryWidth** returns the width in pixels of a null-terminated string in the specified font. The function queries the server for the width computation.

## XQueryWindow

**Status XQueryWindow (***w, info***)**
**Window** *w*;
**WindowInfo** ***info**;

*w*          Window to query.

*info*        Pointer to the following client-supplied structure, filled-in with the appropriate values:

**typedef struct _WindowInfo {**
        **short** *width, height*;
        **short** *x, y*;
        **short** *bdrwidth*;
        **short** *mapped*;
        **short** *type*;
        **Window** *assoc_wind*;
**} WindowInfo;**

The fields in the preceding structure are defined as follows:

| | |
|---|---|
| *width, height* | Size of the window. |
| *x, y* | Coordinates of the window. |
| *bdrwidth* | Width of the window border. |
| *mapped* | Display state of the window. Possible values are: |
| | *IsUnmapped* if the window is unmapped. |
| | *Ismapped* if the window is mapped and displayed (all ancestors also mapped). |
| | *IsInvisible* if the window is mapped but some ancestor is not mapped. |
| *type* | Type of window. Possible values include: |
| | *IsTransparent* if the window is transparent. |
| | *IsOpaque* if the window is a normal opaque window. |
| | *IsIcon* if the window is an icon window. If so, the *assocwind* field contains the icon's corresponding regular window. |
| *assocwind* | Contains the window's icon window, if any. |

**XQueryWindow** gets various facts about a window. The function fills in the client-passed **WindowInfo**, which is defined in **< X/Xlib.h >**.

The procedure returns 0 if unsuccessful, and may fail if the window has been destroyed.

## XRaiseWindow

**XRaiseWindow** (*w*)
**Window** *w*;

w        Window to be raised.

**XRaiseWindow** raises the window so that no sibling window obscures it. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack, while leaving its x and y location on the desk constant.

Raising a mapped opaque window may generate exposure events for the window and any mapped opaque subwindows that were formerly obscured.

Raising a transparent window does not affect the screen. Transparent windows never obscure other windows for the purposes of output, but do obscure for the purposes of cursor and input control.

## XReadBitmapFile

**Status XReadBitmapFile** (*fn, width, height, data, x_hot, y_hot*)
**char** *\*fn*;
**int** *\*width, \*height*;
**short** *\*\*data*;
**int** *\*x_hot, \*y_hot*;

*fn*        Bitmap filename.

*width*        Returned width of the bitmap.

*height*        Returned height of the bitmap.

*data*        Bitmap data.

*x_hot, y_hot*
       Returned coordinates of the hot spot.

**XReadBitmapFile** reads a bitmap from the named file. A bitmap file contains C language source code in the following format:

```
#define xxx_width
#define xxx_height
#define xxx_x_hot
#define xxx_y_hot
static short xxx_bits[ ] = {0xNNNN, 0xNNNN, ... 0xNNNN};
```

where *xxx* is a name usually derived from the name of the file and **NNNN** are hexadecimal digits. *xxx***_bits[ ]** should be initialized with `((xxx_width + 15) >> 4) * xxx_height)` shorts.

The variables that end with **_x_hot** and **_y_hot** are optional and present only if a hotspot has been defined for the bitmap.

If the file cannot be opened, **XReadBitmapFile** returns a Status of 0. If the file can be opened but is not syntactically valid, the procedure returns a negative Status. If the file is readable and valid, it returns a Status of 1.

**XReadBitmapFile** assigns the bitmap's *width* and *heigth*, as read from the file, to the caller's variables *width* and *height*. The function then allocates an appropriate amount of storage, reads the bitmap data from the file, and assigns the caller's variable data. The caller must free *data* when no longer needed.

If *x_hot* and *y_hot* are not NULL, then **XReadBitmapFile** sets *x_hot* and *y_hot* to the value of the hot spot as defined in the file. If no hot spot is defined, **XReadBitmapFile** sets *x_hot* and *y_hot* to -1.


# XRebindCode

**XRebindCode (***keycode, shiftbits, str, nbytes,* **)**
**unsigned int** *keycode*;
**unsigned int** *shiftbits*;
**char \****string*;
**int** *nbytes*;


*keycode*    Keycode to temporarily change.

*shiftbits*   Shift bits.

*str*       Pointer to the string to assign.

*nbytes*    Number of bytes in the string.

If you wish to rebind the keyboard, you can use this routine to change (on a non-permanent basis) the binding of the keyboard. After issuing this rebind function, subsequent calls to **XLookupMapping** returns the supplied string. The string should be stored in static storage; an automatic string may be deallocated by the time it is needed.

If *nbytes* is 0 and *str* is not NULL, then *str* is assumed to point to a two-byte array that contains the code page and code point of a dead key. If *str* is NULL and *nbytes* is not zero, then *nbytes* defines a function ID.

See *AIX Operating System Technical Reference* for more information on function IDs, code pages and code points.

## XRemoveHost

#include < sys/socket.h >

XRemoveHost (*host*)
struct innadr *\*host*;

*host*      Network address of host to remove.

This function removes the specified host from the list of hosts allowed to open connections to the display. The display hardware must be on the same host as the client process. If you remove your machine from the access list, no new connections can be made. There is no way back from this call short of logout.

## XRotateBuffers

XRotateBuffers (*n*)
int *n*;

*n*      Factor by which to rotate buffers.

This function rotates the buffers by $n$. Buffer 0 becomes buffer $n$, buffer 1 becomes $(n+1)$ mod 8, and so on. This buffer numbering is global to the display.

## XScreenSaver

XScreenSaver (*savetimeout, patterntimeout, video*)
int *savetimeout, patterntimeout*;
int *video*;

*savetimeout*
      Number of minutes the server is idle before the screen is blanked by a pattern.
*patterntimeout*
      Number of minutes a pattern is displayed before being changed.
*video*      Determines whether a root tile or background color is used.

If the server remains idle for the specified number of minutes, the server blanks the screen, usually with a pattern that changes at the specified rate. The screen state is typically restored when the next request or input event occurs, unless the server was out of memory. If the server was out of memory, exposure events are generated for all mapped windows. If

*video* is non-zero, the server uses the current background color to blank the screen; otherwise, the root tile is used.

## XSelectInput

**XSelectInput** (*w, mask*)
**Window** *w*;
**int** *mask*;

*w*        Window for which input events are selected.

*mask*     Mask indicating the events the window selects.

**XSelectInput** defines the input events a window is interested in. If a window is not interested in an event, the event is usually propagated up to the closest ancestor that is interested. The bits of the mask are defined in "Events" on page 4-18.

If you select **ExposeRegion**, you also select **ExposeWindow**.

A call to **XSelectInput** overrides any previous call to **XSelectInput** for the same window, whether from the same client or a different one. Two clients cannot each select events simultaneously from the same window. Initially, no events are generated on a window.

If a window has both **ButtonPressed** and **ButtonReleased** selected, then a **ButtonPressed** event in that window automatically grabs the mouse until all buttons are released, with events sent to windows as described for **XGrabMouse**. This ensures that a window sees the release event corresponding to the pressed event, even though the mouse may have exited the window in the meantime.

If **MouseMoved** is selected, events are sent independent of the state of the mouse buttons. Instead, if one or more of **RightDownMotion**, **MiddleDownMotion**, or **LeftDownMotion** is selected, **MouseMoved** events are generated only when one or more of the specified buttons is pressed. (There are no events of type **RightDownMotion**, **MiddleDownMotion**, or **LeftDownMotion**; these are ways to request **MouseMoved** events only when particular buttons are held down).

## XSetDisplay

**XSetDisplay** (*display*)
**Display** *\*display*;

*display*    Pointer to the Display structure to use.

This function sets the current *display* connection to which you are talking and is used to switch between displays. The Display structure is returned by **XOpenDisplay**.

## XSetIconWindow

**XSetIconWindow** (*w, iw*)
**Window** *w*;
**Window** *iw*;

*w*        Window for which to set an icon window.

*iw*       Icon window.

**XSetIconWindow** sets the icon window for a window. The icon window must be a sibling of the specified window, both windows must be opaque, and neither can already be an icon window. When created, windows do not have icon windows defined.

The icon window facility is provided because many window manager programs allow the user to turn a window into an icon. A client should call **XSetIconWindow** to control the contents of the window's icon. If the client has not called **XSetIconWindow**, the window manager should create its own icon window for the window. If a window is destroyed and has a mapped icon window, that icon window is unmapped and receives an **UnmapWindow** event. If a window is destroyed and is a mapped icon window, its corresponding regular window is mapped.

## XSetResizeHint

**XSetResizeHint** (*w, width0, height0,widthinc, heightinc*)
**Window** *w*;
**int** *width0, height0, widthinc, heightinc*;

*w*        Window for which resize hint data is supplied.

*width0, height0*
           Base size of the specified window.

*widthinc, heightinc*
           Increment values by which resize is computed.

**XSetResizeHint** is used to give a hint to the window system that can be used by a window manager program to define the desired shape of a window. The inside height of the window should be the base height (*height0*) plus some multiple of the height increment (*heightinc*), and the inside width of the window should be the base width (*width0*) plus some multiple of the width increment (*widthinc*). These parameters are hints for the window manager. They may or may not be honored.

By default, a window's resize hint is (0, 0, 1, 1).

The base height and width must be non-negative, and the height and width increments must be positive. The increment values usually depend on font sizes for text windows.

This function can be used by window managers to avoid resizing windows to sizes that may not be convenient for the clients, but clients must not presume that the window is the correct size.

## XStippleFill

**XStippleFill** (*w, x, y, width, height, pixel, stipmask, func, planes*)
**Window** *w*;
**int** *x, y, width, height*;
**int** *pixel*;
**Bitmap** *stipmask*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Destination window. |
| *x, y* | Upper-left coordinates of the region to be filled. |
| *width, height* | Width and height of the region to be filled. |
| *pixel* | Source value to use to modify the region. |
| *stipmask* | Stipple mask. |
| *func* | Display function. |
| *planes* | Plane mask. |

This function performs a display function in a region of the window using a repeating pattern defined by the stipple bitmap. The tiling origin is controlled by the window's tile mode. The *pixel* value is used as the source. The destination pixels modified are those corresponding to the 1 values in the pattern generated from the stipple bitmap. A stipple mask is a 16 pixel by 16 pixel bitmap used to tile the region. It serves as an additional clip mask for a fill operation.

## XStoreBitmap

> **Bitmap XStoreBitmap** (*width*, *height*, *data*)
> **int** *width*, *height*;
> **short** *\*data*;

*width, height*   Size of the bitmap to save.

*data*          Pointer to the bitmap.

**XStoreBitmap** creates a bitmap from client-supplied data for later use and returns a Bitmap ID. The client should call **XFreeBitmap** when finished with it.

This function returns 0 if unsuccessful.

## XStoreBuffer

> **XStoreBuffer** (*bytes*, *nbytes*, *buffer*)
> **char** *\*bytes*;
> **int** *nbytes*;
> **int** *buffer*;

*bytes*     Pointer to the bytes to store.

*nbytes*    Number of bytes to store.

*buffer*    Buffer in which to store the bytes.

This function stores an arbitrary string of bytes into the specified buffer. The buffer's contents may be retrieved later by any client calling **XFetchBytes**. Note that the buffer's contents are not necessarily ASCII or null-terminated, so null bytes are not special. There are eight buffers, numbered 0-7.

## XStoreBytes

> **XStoreBytes** (*bytes*, *nbytes*)
> **char** *\*bytes*;
> **int** *nbytes*;

*bytes*     Pointer to the bytes to store.

*nbytes*    Number of bytes to store.

**XStoreBytes** stores an arbitrary string of bytes into buffer number 0. The buffer's contents may be retrieved later by any client calling **XFetchBytes**. Note that the buffer's contents are not necessarily ASCII or null-terminated, so null bytes are not special.

# XStoreColor

XStoreColor (*def*)
Color *\*def*;

*def*        Returned closest available hardware color for the specified pixel.

This function sets the color of the specified pixel value to the closest available hardware color. Note that it must be a read/write cell.

# XStoreColors

XStoreColors (*ncolors, defs*)
int *ncolors*;
Color *\*defs*;

*ncolors*    Number of color definitions.

*def*        Returned closest available hardware color for the specified colors.

This function changes the colors of *ncolors* pixels to the closest available hardware colors. Note that these must be read/write cells.

# XStoreCursor

Cursor XStoreCursor (*cursor, clipmask, xoff, yoff, fg, bg, func*)
Bitmap *cursor*;
Bitmap *clipmask*;
int *xoff, yoff*;
int *fg, bg*;
int *func*;

*cursor*     Bitmap for the two-plane cursor.

*clipmask*   Clip mask

# XStoreCursor

*xoff, yoff*   Coordinates of a point in the bitmap that corresponds to the position of the mouse.

*fg*        Foreground color of the cursor.

*bg*        Background color of the cursor.

*func*     Display function (**GXcopy** or **GXxor**).

This function stores a cursor in the window system. The colors of the cursor are defined by the pixel values *fg* and *bg*. If *mask* is zero, all pixels of the cursor are displayed. The mask bitmap, if present, must be the same size as the cursor bitmap.

The bitmaps can be freed immediately if no further explicit references to them are to be made. The components of the cursor may be transformed arbitrarily to meet hardware limitations.

# XStoreName

**XStoreName (***w, name***)**
**Window** *w*;
**char \****name*;

*w*        Window to be named.

*name*    Pointer to a null-terminated string that contains the name.

**XStoreName** assigns a name to a window. The name should be a null-terminated string which is returned by any subsequent call to **XFetchName**. Windows are typically named for the convenience of window managers. This allows a window manager to display a text representation of a window when its icon is being displayed.

# XStorePixmapXY

**Pixmap XStorePixmapXY (***width, height, data***)**
**int** *width, height*;
**short \****data*;

*width, height*  Size of the pixmap to store.

*data*           Pointer to the pixmap data.

This function creates a pixmap of the specified size from client-supplied data and returns a Pixmap ID. The data must be in the XY format. See "Pixels and Planes" on page 4-15 for details on the format. This data is stored in the window system for later use.

This function returns 0 if the pixmap could not be created. The client should call **XFreePixmap** when finished with the pixmap.

## XStorePixmapZ

**Pixmap XStorePixmapZ** (*width, height, data*)
**int** *width, height*;
**caddr_t** *data*;

*width, height*   Size of the pixmap to store.

*data*   Pointer to the pixmap data.

This function creates a pixmap of the specified size from client-supplied data and returns a pixmap ID. The data must be in the Z format. See "Pixmaps" on page 4-16 for details on the format. This data is stored in the window system for later use.

This function returns 0 if the pixmap could not be created. The client should call **XFreePixmap** when finished with the pixmap.

## XStringWidth

**XStringWidth**(*string, info, charpad, spacepad*)
**char** *\*string*;
**FontInfo** *\*info*;
**int** *charpad, spacepad*;

*string*   Pointer to the string for which a width is computed.

*info*   Pointer to the supplied font info structure. For a definition of the fields in this structure, see "Font" on page 4-27.

*charpad, spacepad*
   Value to be added to the width of each character and space, respectively, defined in the string.

This function computes the width of the string given a complete *FontInfo* structure. The *charpad* and *spacepad* values are added to the width on each character and space defined in the string. This function does not reference the window system server, as the information is all available locally in this case.

## XSync

**XSync** (*discard*)
**int** *discard*;

*discard*    Discard flag, set to 1 for true or 0 for false.

**XSync** flushes the output buffer, then waits until all events and errors resulting from previous calls have been received and processed by the X server. Events are placed on the input queue. The client's **XError** subroutine is called once for each error received.

If *discard* is set to 1, **XSync** discards all events on the input queue, including those events that were on the queue before this function was called.

Few clients will need to use this subroutine. Although it can be useful for debugging, **XSync** has a detrimental effect on performance.

## XText

**XText** (*w, x, y, string, len, font, fg, bg*)
**Window** *w*;
**int** *x, y*;
**int** *len*;
**char** *\*str*;
**Font** *font*;
**int** *fg, bg*;

*w*        Window in which to draw text.

*x, y*     Coordinates of the upper-left corner of the first character to draw.

*string*   Text to draw.

*len*      Number of characters from *string* to draw.

*font*     Font to use when drawing text.

*fg*       Foreground pixel value.

*bg*       Background pixel value.

**XText** draws text into a window, using the specified **font** and display function **GXcopy**. The function modifies all planes of the display memory. This function does no padding.

The number of characters to be drawn must be specified in the *len* parameter; **XText** does not assume that *str* is null-terminated.

For each character drawn, a rectangular bitmap is transferred onto the display. All pixels in a character cell are modified to either the *fg* or *bg* pixel value.

## XTextMask

**XTextMask** (*w, x, y, string, len, font, fg*)
**Window** *w*;
int *x, y*;
int *len*;
char *\*str*;
**Font** *font*;
int *fg*;

*w*          Window in which to draw text.

*x, y*        Coordinates of the upper-left corner of the first character to draw.

*string*      Text to draw.

*len*         Number of characters from *string* to draw.

*font*        Font to use when drawing text.

*fg*          Foreground pixel value.

**XTextMask** draws text into a window, using the specified *font* and display function **GXcopy**. The function modifies all planes of the display, only modifying bits specified by the font. The font bits are used as a mask, so only bits set to one in the font cause pixels to be modified on the display. This function does no padding.

The number of characters to be drawn must be specified in the *len* parameter; this call does not assume that *string* is null-terminated.

The *x* and *y* coordinates represent the upper left corner of the first character.

## XTextMaskPad

**XTextMaskPad** (*w, x, y, string, len, font, charpad, spacepad, fg, func, planes*)
**Window** *w*;
**int** *x, y*;
**int** *len*;
**char** *\*str*;
**Font** *font*;
**int** *charpad, spacepad*;
**int** *fg*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Window in which to draw text. |
| *x, y* | Coordinates of the upper-left corner of the first character to draw. |
| *string* | Text to draw. |
| *len* | Number of characters from *string* to draw. |
| *font* | Font to use when drawing text. |
| *charpad* | Defines the amount of space to leave between each character. |
| *spacepad* | Defines how much additional padding will occur when a space character is painted. |
| *fg* | Foreground pixel value. |
| *func* | Display function. |
| *planes* | Plane mask. |

**XTextMaskPad** draws text into a window, using the specified *font* and display function *func*. The function modifies the specified planes of the display, only modifying bits specified by the font. The font bits are used as a mask, so only bits set to one in the font cause pixels to be modified on the display.

The number of characters to be drawn must be specified in the *len* parameter; this call does not assume that *string* is null-terminated.

The *x* and *y* coordinates represent the upper left corner of the first character.

*charpad* and *spacepad* can be used for intercharacter and space padding. Padded pixels are not modified.

## XTextPad

**XTextPad** (*w*, *x*, *y*, *string*, *len*, *font*, *charpad*, *spacepad*, *fg*, *bg*, *func*, *planes*)
**Window** *w*;
**int** *x*, *y*;
**int** *len*;
**char** *\*str*;
**Font** *font*;
**int** *charpad*, *spacepad*;
**int** *fg*, *bg*;
**int** *func*;
**int** *planes*;

| | |
|---|---|
| *w* | Window in which to draw text. |
| *x, y* | Coordinates of the upper-left corner of the first character to draw. |
| *string* | Text to draw. |
| *len* | Number of characters from *string* to draw. |
| *font* | Font to use when drawing text. |
| *charpad* | Defines the amount of space to leave between each character. |
| *spacepad* | Defines how much additional padding will occur when a space character is painted. |
| *fg* | Foreground pixel value. |
| *bg* | Background pixel value. |
| *func* | Display function. |
| *planes* | Plane mask. |

**XTextPad** draws text into a window, using the specified **font** and display function. The function modifies the specified planes of the display memory. The number of characters to be drawn must be specified in the *len* parameter; **XTextPad** does not assume that *str* is null-terminated.

For each character drawn, a rectangular bitmap is transferred onto the display.

The character padding *charpad* defines how much space is left between each character. The space padding *spacepad* defines how much additional padding occurs when a space character is painted. Padded pixels are not modified. All pixels in a character cell are modified to either the *fg* or *bg* pixel value.

## XTileAbsolute

**XTileAbsolute** (*w*)
**Window** *w*;

*w*          Window for which to set absolute tile mode.

This function sets the tile mode of the window. In absolute mode (the normal case for opaque windows), tiles are laid out with the upper left corner of the window as the effective origin. The tile mode affects painting of the background for exposures and for **XClear**, **XTileFill**, and **XDrawFilled** requests.

This function does not change the current contents of the window, and you may wish to clear and repaint the screen after this function completes.

## XTileFill

**XTileFill** (*w, x, y, width, height, tile, clipmask, func, planes*)
**Window** *w*;
int *x, y, width, height*;
**Pixmap** *tile*;
**Bitmap** *clipmask*;
int *func*;
int *planes*;

*w*          Destination window.

*x, y*       Upper-left coordinates of the region to be filled.

*width, height*
             Width and height of the region to be filled.

*tile*       Source pixmap that specifies the pattern to use to fill the region.

*clipmask*   Clip mask.

*func*       Display function.

*planes*     Plane mask.

**XTileFill** performs a display function in a region of the window using a repeating pattern defined by the *tile* pixmap. The tiling origin is controlled by the window's **tilemode**. If a clipmask is specified, it defines which pixels of the destination will be affected, and it must be the same height and width as the destination region.

## XTileRelative

**XTileRealtive** (*w*)
**Window** *w*;

*w*          Window for which to set relative tile mode.

This function sets the tile mode of the window. In relative mode (the default for transparent windows), tiles are laid out with the upper left corner of the closest parent window with an absolute tile mode as an effective origin. The tile mode affects painting of the background for exposures and for **XClear**, **XTileFill**, and **XDrawFilled** requests.

This function does not change the current contents of the window, and you may wish to clear and repaint the screen after this function completes.

## XTileSet

**XTileSet** (*w, x, y, width, height, tile*)
**Window** *w*;
**int** *x, y, width, height*;
**Pixmap** *tile*;

*w*          Destination window.

*x, y*       Upper-left coordinates of the region to be filled.

*width, height*
             Width and height of the region to be filled.

*tile*       Source pixmap that specifies the pattern to use to set the region.

**XTileSet** performs a display function in a region of the window using a repeating pattern defined by the *tile* pixmap. The tiling origin is controlled by the window's **tilemode**.

**XTileSet** defaults to modifying all planes of the display with **GXcopy**. No clipping mask is used.

## XUndefineCursor

> **XUndefineCursor** (*w*)
> **Window** *w*;

*w*          Window for which a cursor definition is removed.

**XUndefineCursor** removes the definition of a cursor made by **XDefineCursor** for this window. When the mouse is in the window, the parent's cursor is then used.

On the root window, with no cursor specified, the default cursor is restored.

## XUngrabButton

> **XUngrabButton** (*mask*)
> **int** *mask*;

*mask*      Button mask bits.

**XUngrabButton** notifies the server that the client is no longer interested in grabbing the mouse when the specified button/key combination occurs. This grab is overridden by a grab mouse request.

The mask must have exactly one of the **LeftMask, MiddleMask,** and **RightMask** bits set, and may have some combination of the **AltGraphMask, ControlMask, MetaMask, ShiftLockMask,** and **ShiftMask** bits set as well.

## XUngrabMouse

> **XUngrabMouse** ( )

**XUngrabMouse** releases hold of the mouse if it was grabbed by **XGrabMouse**.

## XUngrabServer

XUngrabServer ( );

This request releases hold of the server if the server was grabbed by **XGrabServer**.

## XUnmapSubwindows

XUnmapSubwindows ($w$)
Window $w$;

$w$        Window whose subwindows are to be unmapped.

This function unmaps all subwindows of the specified window, generates an **UnmapWindow** event on each subwindow, and generates **Exposure** events on formerly obscured opaque windows.

This technique is more efficient than unmapping many windows one at a time, because much of the work of unampping windows can be done once for all windows rather than once for each window.

## XUnmapTransparent

XUnmapTransparent ($w$)
Window $w$;

$w$        Window whose transparent is to be unmapped.

This function unmaps the window but does not affect the screen (even if the window is opaque) and does not generate any exposure (or unmap) events. This function is intended for use mainly by pop-up menus in conjunction with **XPixmapSave** to suppress exposure events. The client should normally restore the saved pixmap to the area formerly covered by the unmapped window.

## XUnmapWindow

**XUnmapWindow** (*w*)
**Window** *w*;

*w*          Window to unmap.

**XUnmapWindow** unmaps the specified window. Any child window will no longer be visible until another map call is made on the parent. That is, the subwindows are still mapped, but not visible until the parent is mapped. This function generates an **UnmapWindow** event for *w*, regardless of whether it is opaque or transparent. Child windows will not receive **UnmapWindow** events.

Unmapping a transparent window does not affect the screen or generate any exposure events. Unmapping an opaque window will generate exposure events on opaque windows that were formerly obscured by it and its children.


## XUpdateMouse

**Status XUpdateMouse** (*w, x, y, subw*)
**Window** *w*;
**int** \**x*, \**y*;
**Window** \**subw*;

*w*          Window in which to update mouse.

*x, y*       Returned coordinates of the mouse relative to the window's top left inside corner.

*subw*       If the mouse is also in a child window, this argument is set to that child; otherwise, *subw* is set to 0.

**XUpdateMouse** is similar **XQueryMouse**, but this function also reads pending events and eliminates any MouseMoved events at the head of the queue. A good way to track the mouse is to use a MouseMoved event as a hint by calling this routine to get up-to-date coordinates.

## XUseKeymap

**Status XUseKeymap (***keymapfile***)**
**char ***keymapfile**;

*keymapfile* Name of the keymap file to use within the current process.

If you wish to use an alternate keymap file, you can use this routine to change the file used. This change only affects the keymap within the current process. The function returns a 0 if it cannot find the keymap file named by **keymapfile** or if the file contains a bad magic number. If the function fails, the existing keymap is untouched.

## XWarpMouse

**XWarpMouse (***w, x, y***)**
**Window ***w***;**
**int ***x, y***;**

*w*          Window to which to move the mouse.

*x, y*       Coordinates relative to top inside-left corner of the window to which the mouse is moved.

**XWarpMouse** moves the mouse to the specified position in the specified window. The *x* and *y* coordinates are relative to the top left-inside corner of the window.

## XWindowEvent

```
XWindowEvent (w, mask, rep)
Window w;
int mask;
XEvent *rep;
```

w           Window from which to search for specific events.

mask        Defines the events for which to search.

rep         Pointer to an **XEvent**.

This function is used to look for specific events from specific windows. **XWindowEvent** flushes the output buffer, then removes the next event in the queue that matches both the passed window and the passed mask. The event is copied into an **XEvent** supplied by the caller. Events earlier in the queue are not discarded. If no such event has been queued, **XWindowEvent** blocks until one is received.

# IBM-Specific X-Windows Implementation

IBM's X-Windows program differs from MIT's X version 10.4 specification as follows:

- Unsupported functions

  The following functions from version 10.4 of X are not supported on the RT PC, and, if used, they simply return:

  **XAddNode**
  **XGetNode**
  **XRemoveNode**

- Additional functions

  The following functions have been added by IBM:

**XKillClient (*w*)**
**Window *w*;**

    *w*       Window owned by the client to kill.

This function closes down the connection to the client that owns the specified window and frees any other resources owned by that client. Clients that lose the connection to the server usually terminate. Upon successful completion, a value of 1 is returned; a value of 0 is returned for unsuccessful completion. This function also returns 0 if the X Server is not an IBM RT PC X-Windows server.

**XQuerySetOptions (*opt*)**
**SetOptions *\*opt*;**

    *opt*      Pointer to the returned set options structure (defined in **X/Xlib.h**).

This function provides access to the terminal characteristics set during X initialization. The returned set options structure (defined in **X/Xlib.h**) contains information on the keyboard settings (bell volume, key click volume, auto-repeat mode, and key lock mode), mouse motion settings (acceleration and threshold), and timing intervals (wait time and long time). Upon successful completion, a value of 1 is returned; a value of 0 is returned if unsuccessful. This function also returns 0 if the X Server is not an IBM RT PC X-Windows server.

**XRevShorts (*data, count*)**
**short *\*data*;**
**int *count*;**

    *data*    Pointer to an array of shorts.

    *count*   Number of shorts.

This function reverses the bits in short (16-bit) words. For example, in the X-Windows environment, the most significant bit is bit 15 of the word and the least significant bit is bit 0 of the word. This function simply reverses the bit ordering of the specified word or words.

- Additional macros

  The following macros have been added by IBM. Do not use these macros before the first call to **XOpenDisplay**:

  **DisplayHeightMm( )** – Provides the screen height in millimeters
  **DisplayWidthMm( )** – Provides the screen width in millimeters
  **DisplayModel( )** – Provides the model number of the display
  **DisplayVrmId( )** – Provides the VRM identification number of the display
  **DisplayOrganization( )** – Provides the display organization by plane or pixel
  **IBM_Server** – Flag indicating attachment to an IBM server.
  **KeybdId( )** – Identifies the keyboard hardware:
      **XDEV_IBM_K101** – U.S. English keyboard (101 key)
      **XDEV_IBM_K102** – European keyboard (102 key)
      **XDEV_IBM_K106** – Japanese keyboard (101 key).

- Font structure

  In the IBM implementation of the font information structure (**FontInfo**), *firstchar* and *lastchar* have been changed to shorts to accommodate fonts with more than 256 characters.

- Time within detail events

  Time values are maintained only for button events, not for key events.

- State bits in key detail

  IBM uses an additional key state, the **Alt-Graphics** key state, and this state is reflected in the high-order byte of the key detail information.

# Sample X-Windows Program

The following program (**xrefresh**) is a relatively simple X-Windows application. This program may be useful if the contents of your screen have been corrupted by a program error (such as unintentionally using the **RootWindow**) or by messages put out by the system underneath your windows.

```
#include <X/Xlib.h>
#include <stdio.h>

main (argc, argv)
int argc;
char **argv;
{
  Window w;

  if(XOpenDisplay(argc ? argv[1] : " ") = = NULL)
          fprintf(stderr, "Could not open Display\n");
  w = XCreateWindow(RootWindow, 0, 0, DisplayWidth( ), DisplayHeight( ),
          0, (Pixmap) 0, (Pixmap) 0);
  XMapWindow(w);     /* Put it on the screen */
  XDestroyWindow(w); /* Throw it away*/

  XFlush( );                       /* Make sure the server sees it */
}
```

This program basically connects to the display, creates a window with a black background and zero-width border over the root window, maps the window to the screen, and then destroys it. This results in exposure events sent to the client programs that have selected exposure events on all mapped unobscured windows. This causes most clients to repaint their windows. The call to **XFlush** is necessary to flush the output buffer because no input call occurs after **XDestroyWindow**. Failure to flush the output buffer is a common programming error in the X-Windows environment. The background pixmap is 0, so the window is covered with the background pixmap.

# Chapter 5.  X-Windows Technical Reference

# CONTENTS

# xterm HFT Functions

VTLs are supported, with following characteristics:

- Locator reports are given for the following mouse events:

  - ButtonPressed
  - ButtonReleased
  - RightDownMotion
  - LeftDownMotion
  - MiddleDownMotion

- Locator reports are given in absolute coordinates.

- Mouse motion is processed in compressed mode.

- Applications need not display a mouse cursor.

A font table is maintained for each invocation of **xterm** to support the change font VTD.

The file **/usr/lpp/X/defaults/Xfonts** identifies the fonts configured on the RT PC. This file controls the mapping of font IDs to font files recognized by **xterm**. A request from a program to **xterm** to query font VTD returns the font ID based on the contents of the **Xfonts** file. The change font VTD changes the local **xterm** font table according to the IDs defined in the **Xfonts** file.

The format of each line in the **Xfonts** file is:

*id   path   style   attr   width   height*

The lines in the file that start with # are ignored. The fields within a line must be separated by one or more blanks.

The parts of each line in **Xfonts** indicate the following:

| | |
|---|---|
| *id* | Specifies the ID to be associated with a particular font file. The decimal values 0 through 65536 are valid. Other programs can add to this file once their fonts match the common font file format. |
| *path* | Specifies the name of the font file to be opened. The font file must be in the directory **/usr/lpp/fonts**. |
| *style* | Specifies a number identifying the style of the font. |
| *attr* | Specifies a number identifying the attributes of the font. The defined attributes are:<br><br>• **0** (HFFNTPLAIN)<br>• **1** (HFFNTBOLD)<br>• **2** (HFFNTITALIC)<br><br>Refer to the file **/usr/include/sys/hft.h** for definitions of HFFNTPLAIN, HFFNTBOLD, and HFFNTITALIC. |
| *width* | Specifies the width of the font characters in pels. |
| *height* | Specifies the height of the font characters in pels. |

For more information, see "hft" and "remote processing" in the *AIX Operating System Technical Reference*.

| Function | HFT | xterm |
|---|---|---|
| TERMINAL CONTROL | | |
| Query I/O Error | ioctl | no support |
| Query Device | ioctl | no support |
| Reconfigure | ioctl | no support |
| Get Channel Number | ioctl | no support |
| Set Echo and Break Map | ioctl | no support |
| Set Keyboard Map | ioctl | hftctl |
| Get Virtual Terminal ID | ioctl | no support |
| Query Device ID | ioctl | no support |
| Query Physical Device (ID = 0) | ioctl | hftctl |
| Query HFT device | ioctl | hftctl |
| Query Locator | ioctl | hftctl |
| Query LPFKs | ioctl | no support |
| Query Dials | ioctl | no support |
| Query PS | ioctl | hftctl |
| Enable / Disable Sound | ioctl | no support |
| Enter / Exit Monitor Md | ioctl | no support |
| Query Screen Manager | ioctl | no support |
| Control Screen Manager | ioctl | no support |
| KSR Protocol | VTD | VTD |
| Chararcter Set Definition | VTD | no support |
| Set KSR Color Palette | VTD | VTD |
| Change Fonts | VTD | VTD |
| Cursor Representation | VTD | VTD |
| INPUT | | |
| Dead Key Support | read | supported |
| Code Page Shift | read | supported |

| Function | HFT | xterm |
|---|---|---|
| Untranslated Key | read | supported |
| Input Device Report | read | supported |
| mouse | VTL | supported |
| tablet | VTL | no support |
| dials | VTL | no support |
| Lighted Program Function Keys | VTL | no support |
| Sound Complete | signal | no support |
| OUTPUT | | |
| Write ASCII (Code Page 850) | write | supported |
| Write Code Page Shift | write | supported |
| Set LEDs | VTD | no support |
| Set Locator Threshold | VTD | no support |
| Set Table Dead Zone | VTD | no support |
| Set LPFKs | VTD | no support |
| Set Dials | VTD | no support |
| Sound output | VTD | no support |
| Cancel Sound | VTD | no support |
| Change Physical Display | VTD | no support |

# xterm Datastream Support

The following is a list of the escape sequences supported by **xterm**.

Some escape sequences activate and deactivate an alternate screen buffer that is the same size as the display area of the window. This capability allows the contents of the screen to be saved and restored. When the alternate screen is activated, the current screen is saved and replaced with the alternate screen. The saving of lines scrolled off of the window is disabled until the original screen is restored.

This table uses these abbreviations in the righthand column:

Xv   Supported by xterm running in vt100 mode.
Xh   Supported by xterm running in hft mode.
H    Found in hft datastream.
V    Found in vt100 datastream.

| Name | Function | Datastream | Support |
|------|----------|------------|---------|
|      | SINGLE-BYTE CONTROLS | | |
| BEL | Bell | 0x07 | Xv, Xh, H, V |
| BS | Backspace | 0x08 | Xv, Xh, H, V |
| HT | Horizontal tab | 0x09 | Xv, Xh, H, V |
| LF | Linefeed | 0x0A | Xv, Xh, H, V |
| VT | Vertical tab | 0x0B | Xv, Xh, H, V |
| FF | Form feed | 0x0C | Xv, Xh, H, V |
| CR | Carriage return | 0x0D | Xv, Xh, H, V |
| SO | Shift out | 0x0E | Xv, Xh, H, V |
| SI | Shift in | 0x0F | Xv, Xh, H, V |
| DC1 | Device control 1 | 0x11 | H, V |
| DC3 | Device control 3 | 0x13 | H, V |
| CAN | Cancel | 0x18 | H, V |

| Name | Function | Datastream | Support |
|---|---|---|---|
| SUB | Substitute (also cancels) | 0x1A | H, V |
| ESC | Escape | 0x1B | Xv, Xh, H, V |
| SS4 | Single Shift 4 | 0x1C | Xh, H |
| SS3 | Single Shift 3 | 0x1D | Xh, H |
| SS2 | Single Shift 2 | 0x1E | Xh, H |
| SS1 | Single Shift 1 | 0x1F | Xh, H |
| cbt | cursor back tab | ESC [ Pn Z | H |
| cha | cursor horizontal absolute | ESC [ Pn G | H |
| cht | cursor horizontal tab | ESC [ Pn I | H |
| ctc | cursor tab stop control | ESC [ Pn W | H |
| cnl | cursor next line | ESC [ Pn E | H |
| cpl | cursor preceding line | ESC [ Pn F | Xv, Xh, H |
| cpr | cursor position report | ESC [ Pl; Pc R | Xv, Xh, H, V |
| cub | cursor backward | ESC [ Pn D | Xv, Xh, H, V |
| cud | cursor down | ESC [ Pn B | Xv, Xh, H, V |
| cuf | cursor forward | ESC [ Pn C | Xv, Xh, H, V |
| cup | cursor position | ESC [ Pl; Pc H | Xv, Xh, H, V |
| cuu | cursor up | ESC [ Pn A | Xv, Xh, H, V |
| cvt | cursor vertical tab | ESC [ Pn Y | H |
| da1 | DEVICE ATTRIBUTES<br>request (host to vt100)<br>request (host to vt100)<br>response (vt100 to host) | <br>ESC [ c<br>ESC [ 0 c<br>ESC [ ? 1 ; 2 c | <br>Xv, Xh, V<br>Xv, Xh, V<br>Xv, Xh, V |
| dch | delete character | ESC [ Pn P | Xv, Xh, H |
| decaln | screen alignment display | ESC # 8 | Xv, Xh, V |

| Name | Function | Datastream | Support |
|---|---|---|---|
| deckpam | keypad application mode | ESC = | Xv, V |
| deckpnm | keypad numeric mode | ESC > | Xv, V |
| decrc | restore cursor & attributes | ESC 8 | Xv, Xh, V |
| decsc | save cursor & attributes | ESC 7 | Xv, Xh, V |
| decstbm | set top & bottom margins | ESC [ Pt; Pb r | Xv, Xh, V |
| dl | delete line | ESC [ Pn M | Xv, Xh, H |
| dsr | device status report<br>0 response from vt100:  ready<br>5 command from host:  please report status<br>6 command from host:  report active position<br>13 error report sent from virtual terminal to host | ESC [ Ps n | <br>Xv, Xh, V<br>Xv, Xh, V<br><br>Xv, Xh, H, V<br><br>H |
| dmi | disable manual input | ESC ` (back quote) | H |
| emi | enable manual input | ESC b | H |
| ea | erase area<br>0 erase to end of area<br>1 erase from area start<br>2 erase all of area | ESC [ Ps O | <br>Xv, Xh, H<br>Xv, Xh, H<br>Xv, Xh, H |
| ed | erase display<br>0 erase to end of display<br>1 erase from display star<br>2 erase all of display | ESC [ Ps J | <br>Xv, Xh, H, V<br>Xv, Xh, H, V<br>Xv, Xh, H, V |
| ef | erase field-e,s,all<br>0 erase to end of field<br>1 erase from field start<br>2 erase all of field | ESC [ Ps N | <br>Xv, Xh, H<br>Xv, Xh, H<br>Xv, Xh, H |
| el | erase line<br>0 erase to end of line<br>1 erase from start of line<br>2 erase all of line | ESC [ Ps K | <br>Xv, Xh, H, V<br>Xv, Xh, H, V<br>Xv, Xh, H, V |

| Name | Function | Datastream | Support |
|------|----------|------------|---------|
| ech | erase character | ESC [ Pn X | Xv, Xh, H |
| gsm | graphic size modify | ESC [ Pn; Pn Sp B | H |
| hts | horizontal tab stop | ESC H | Xv, Xh, H, V |
| hvp | horizontal and vertical position | ESC [ Pl; Pc f | Xv, Xh, H, V |
| ich | insert character | ESC [ Pn @ | Xv, Xh, H |
| il | insert line | ESC [ Pn L | Xv, Xh, H |
| ind | index | ESC D | Xv, Xh, H, V |
| ls2 | lock shift G2 | ESC n | Xv |
| ls3 | lock shift G3 | ESC o | Xv |
| nel | next line | ESC E | Xv, Xh, H, V |
| ksi | keyboard status information | ESC [ Ps p | H |
| pfk | PF key report | ESC [ Pn q | Xh, H |
| rcp | restore cursor position | ESC [ u | Xv, Xh, H |
| ri | reverse index | ESC M | Xv, Xh, H, V |
| ris | reset to initial state | ESC c | Xv, Xh, H, V |
| rm | reset mode, ANSI specified modes:  SEE set mode (below) | ESC [ Ps;...;Ps l | |
| | reset mode, other private modes and XTERM private modes: SEE set mode (below) | ESC [ ? Ps;...;Ps l | |
| | restore mode, other private modes and XTERM private modes:  SEE set mode (below) | ESC [ ? Ps;...;Ps r | |
| | save mode, other private modes and XTERM private modes: SEE set mode (below) | ESC [ ? Ps;...;Ps s | |
| scp | save cursor postion | ESC [ s | Xv, Xh, H |

| Name | Function | Datastream | Support |
|------|----------|------------|---------|
| scs | select character set | | |
| | United Kingdom Set | ESC ( A    (GO)<br>ESC ) A    (G1)<br>ESC * A    (G2)<br>ESC + A    (G3) | Xv, V<br>Xv, V<br>Xv, V<br>Xv, V |
| | ASCII Set (USASCII) | ESC ( B    (G0)<br>ESC ) B    (G1)<br>ESC * B    (G2)<br>ESC + B    (G3) | Xv, V<br>Xv, V<br>Xv, V<br>Xv, V |
| | special graphics | ESC ( 0    (G0)<br>ESC ) 0    (G1)<br>ESC * 0    (G2)<br>ESC + 0    (G3) | Xv, V<br>Xv, V<br>Xv, V<br>Xv, V |
| sd | scroll down | ESC [ Pn T | H |
| sl | scroll left | ESC [ Pn Sp @ | H |
| sr | scroll right | ESC [ Pn Sp A | H |
| ss2 | single shift G2 | ESC N | Xv |
| ss3 | single shift G3 | ESC O | Xv |
| su | scroll up | ESC [ Pn S | Xv, Xh, H |
| sgr | set graphic rendition<br>    0 normal<br>    1 bold<br>    4 underscore<br>    5 blink (appears as bold)<br>    7 reverse<br>    8 invisible<br>    10..17 fonts<br>    30..37 foreground colors<br>    40..47 background colors<br>    90..97 foreground colors<br>    100..107 background colors | ESC [ Ps m | <br>Xv, Xh, H, V<br>Xv, Xh, H, V<br>Xv, Xh, H, V<br>Xv, Xh, H, V<br>Xv, Xh, H, V<br>Xh, H<br>Xh, H<br>Xh, H<br>Xh, H<br>Xh, H<br>Xh, H |

| Name | Function | Datastream | Support |
|------|----------|------------|---------|
| sg0a | set GO character set | ESC ( f | Xh, H |
| sg0b | set GO character set-ALT | ESC , f | H |
| sg1a | set G1 character set | ESC ) f | Xh, H |
| sg1b | set G1 character set-ALT, where f := {:; < = > ?@} | ESC - f | H |
| sm | set mode | | |
| | ANSI specified modes<br>  4 IRM  insert mode<br>12 SRM  send/rec mode<br>18 TSM  tab stop mode<br>20 LNM  linefeed/newline | ESC [ Ps;...;Ps h | Xv, Xh, H<br>H<br>H<br>Xv, Xh, H, V |
| | Other private modes<br>1 normal/application cursor<br>3 80/132 columns<br>4 smooth/jump scroll<br>5 reverse/normal video<br>6 origin/normal<br>7 on/off autowrap<br>8 on/off autorept<br>21 CNM    CR-NL | ESC [ ? Ps;...;Ps h | Xv, V<br>Xv, Xh, V<br>Xv, Xh, V<br>Xv, Xh, V<br>Xv, Xh, V<br>Xv, Xh, H, V<br>Xv, Xh, V<br>H |
| | XTERM private modes<br>40 132/80 column mode<br>41 curses(5) fix<br>42 hide/show scrollbar<br>43 on/off save scroll text<br>44 on/off margin bell<br>45 on/off reverse wraparound<br>46 start/stop logging<br>47 alternate/normal screen buffer<br>48 reverse/normal status line<br>49 page/normal scroll mode | | Xv, Xh<br>Xv, Xh<br>Xv, Xh<br>Xv, Xh<br>Xv, Xh<br>Xv, Xh<br>Xv, Xh<br>Xv, Xh<br><br>Xv, Xh<br>Xv, Xh |

| Name | Function | Datastream | Support |
|------|----------|------------|---------|
| tbc | tabulation clear<br>0 clear horizontal tab stop at active position<br>1 vertical tab at line indicated by cursor<br>2 horizontal tabs on line<br>3 all horizontal tabs<br>4 all vertical tabs | ESC [ Ps g (default Ps = 0) | Xv, Xh, H, V<br><br>H<br><br>H<br>Xv, Xh, H, V<br>H |
| VTD | virtual terminal data | ESC [ x | Xv, Xh, H |
| VTL | virtual terminal device input | ESC [ y | Xh, H |
| VTR | vt raw keyboard input | ESC [ w | Xh, H |
| vts | vertical tab stop | ESC I | H |
| xes | erase status line | ESC [ ? E | Xv, Xh |
| xrs | return from status line | ESC [ ? F | Xv, Xh |
| xhs | hide status line | ESC [ ? H | Xv, Xh |
| xss | show status line | ESC [ ? S | Xv, Xh |
| xgs | go to column of status line | ESC [ ? Ps T | Xv, Xh |
| xst | set text parameters<br>0 change window name and title to Pt<br>46 change log file to Pt | ESC ] Ps ; Pt \007 | Xv, Xh<br>Xv, Xh<br><br>Xv, Xh |

# X-Windows Protocol

You can build an X-Windows protocol on top of any reliable byte stream. It uses a simple block protocol on top of the stream layer. Most requests are 24-byte blocks; some carry additional data. Requests do not have a corresponding reply unless actual data is expected in return. Error responses are generally not caught by a failing request but by a later request that expects a reply.

On each system, displays are numbered from zero.

For TCP connections, display number N is associated with port 5800 + N (hex 0x58) and with port 5900 + N. The X Server receives and accepts new connections on these ports. The X Server treats connections made on the 58xx port as connections with hosts transmitting data in little indian order (low order byte first). It treats connections made on the 59xx port as connections with hosts transmitting data in big indian order (high order byte first). For connections with opposite-order machines, the X Server swaps bytes for 16-bit and 32-bit quantities in requests and replies, with the exception of host addresses, which should be transmitted in standard network order. Clients do not need to swap bytes, and hosts with the same byte ordering do not need to swap bytes.

The following is a list of some of the C definitions used:

```
typedef long Window;
typedef long Font;
typedef long Bitmap;
typedef long Pixmap;
typedef long Cursor;
typedef long Locator;
```

For the convenience of certain programming languages, the top three bits of **Window**, **Font**, **Bitmap**, **Pixmap**, and **Cursor** values are zero.

A Bitmap is a single plane (bit) rectangle. A Pixmap is an N-plane (pixel) rectangle, where N is the number of planes provided by the particular display. In this protocol, N ranges from 1 to 16. Cursors are used as mouse pointers; a cursor is an arbitrary two-color shape with an arbitrary point. A Locator is an absolute point on the display, represented as < x,y > with the X-coordinate in the high 16 bits and the Y-coordinate in the low 16 bits.

A Bitmap is represented by ((width + 15) / 16) * height * 2 bytes of data. The bits are in scanline order, with each scanline padded if necessary to a multiple of 16 bits. The pad bits are of arbitrary value. Within a scanline, the bits are represented left to right, stored in 16-bit words, but the bits are reversed within a given 16-bit word; the leftmost bit of the scanline is the least significant bit of the word.

A Pixmap can be represented in either XY-format or Z-format. In XY-format, each plane is represented as a Bitmap, and the planes appear from most to least significant bit order. The total number of bytes is thus ((width + 15) / 16) * height * 2 * depth. In Z-format, the pixels are in scanline order, left to right within a scanline. For displays with 2 to 8 planes, each pixel is represented by a single byte; the total number of bytes is width * height. For

displays with 9 to 16 planes, each pixel is represented by a 16-bit word; the total number of bytes is 2 * width * height. Z-format cannot be used on monochrome displays.

Pixel values 0 and 1 are always defined on every display. They are primarily intended for use in monochrome applications. On monochrome displays, pixel value 0 is black and pixel value 1 is white. On color displays, you can redefine the colors.

Display function codes used in all output requests, and their affect on the destination as a function of the source and the destination (NOT binds tightest) are:

| | | |
|---|---|---|
| GXclear | 0x0 | 0 |
| GXand | 0x1 | src AND dst |
| GXandReverse | 0x2 | src AND NOT dst |
| GXcopy | 0x3 | src |
| GXandInverted | 0x4 | NOT src AND dst |
| GXnoop | 0x5 | dst |
| GXxor | 0x6 | src XOR dst |
| GXor | 0x7 | src OR dst |
| GXnor | 0x8 | NOT src AND NOT dst |
| GXequiv | 0x9 | NOT src XOR dst |
| GXinvert | 0xa | NOT dst |
| GXorReverse | 0xb | src OR NOT dst |
| GXcopyInverted | 0xc | NOT src |
| GXorInverted | 0xd | NOT src OR dst |
| GXnand | 0xe | NOT src OR NOT dst |
| GXset | 0xf | 1 |

Given a source and destination pixel, a display function is computed bitwise on corresponding bits of the pixels. That is, a Boolean operation is performed on each bit plane of the display. It uses a plane mask to restrict output operations to a subset of planes; the mask contains a one bit for each affected pixel bit (plane).

The following is an example of a C definition of a request:

```
typedef struct _XReq {
    unsigned char code;
    unsigned char func;
    unsigned short mask;
    Window windowId;
    union {
        long l[4];
        short s[8];
        unsigned short u[8];
        char b[16];
    } param;
} XReq;
```

**Note:** Requests carrying more than 128K-bytes of additional data may cause some of the server implementations to close the client's connection.

Input events (keyboard, mouse button, mouse motion, window change) are generated asynchronously by X and are reported on the same network connection. Clients must therefore expect any number of input events between any two replies to requests. Data coming from X is bundled into Data coming from X is bundled into 24-byte blocks, with a code indicating the type of data. The following is the C definition of this structure:

```
typedef struct _XRep {
    long code;
    union {
        long l[5];
        short s[10];
        unsigned short u[10];
        char b[20];
    } param;
} XRep;
```

The possible **XRep** codes are:

| | | |
|---|---|---|
| **X_Reply** | 0 | Normal reply |
| **X_Error** | -1 | Error |
| **else** | | Asynchronous input event |

The contents of event structures are discussed later. The contents of X_Reply structures vary with each request, and are documented for each such request. The following is the contents of an X_Error structure:

| **code** | **X_Error** |
|---|---|
| **param.l[0]** | number of failing request |
| **param.b[4]** | error code |
| **param.b[5]** | original request code |
| **param.b[6]** | original request function |
| **param.l[2]** | original request window ID |

Requests are counted per network connection, starting from one. The possible error codes are:

| | | |
|---|---|---|
| BadRequest | 1 | bad request code |
| BadValue | 2 | int parameter out of range |

| | | |
|---|---|---|
| BadWindow | 3 | parameter not a Window |
| BadPixmap | 4 | parameter not a Pixmap |
| BadBitmap | 5 | parameter not a Bitmap |
| BadCursor | 6 | parameter not a Cursor |
| BadFont | 7 | parameter not a Font |
| BadMatch | 8 | parameter mismatch |
| BadTile | 9 | Pixmap shape invalid for tiling |
| BadGrab | 10 | mouse/button already grabbed |
| BadAccess | 11 | access control violation |
| BadAlloc | 12 | insufficient resources |
| BadColor | 13 | no such color |

Not all displays support all variations of all output requests. Output requests may be transformed arbitrarily (including being ignored), without errors being generated.

Some displays have very limited memory for storage of off-screen resources such as bitmaps, pixmaps, and fonts.

All requests and replies are padded as necessary to be a multiple of four bytes long. The pad bytes are of arbitrary value.

# X Server Protocol Requests

The protocol requests are listed in order by code.

## X_CreateWindow

| | |
|---|---|
| code | 1 |
| func | border width |
| windowId | parent Window |
| param.s[0] | inside height (not including borders) (> 0) |
| param.s[1] | inside width (not including borders) (> 0) |
| param.s[2] | outer left coordinate (start of border) |
| param.s[3] | outer top coordinate (start of border) |
| param.l[2] | border tile Pixmap or 0 |
| param.l[3] | background tile Pixmap or 0 |

Creates an unmapped (not displayed) opaque window. Coordinates are relative to the inside of the parent. A border pixmap need not be given if the border width is zero. If no background pixmap is given, the parent's background pixmap is used. The tilemode of the new window is **TileModeAbsolute**. The clipmode of the new window is **ClipModeClipped**. The window does not have a defined cursor.

The parent window must be an opaque window.

A window is wholly contained within its parent; that is, any parts of the window that extend outside the parent window are not displayed.

The background and border pixmaps may be freed immediately if no further explicit references to them are to be made.

The reply:

| | |
|---|---|
| param.l[0] | Window |

Errors: **BadWindow**, **BadValue**, **BadPixmap**, **BadTile**, **BadMatch**

## X_CreateTransparency

| | |
|---|---|
| code | 2 |
| windowId | parent Window |
| param.s[0] | inside height (> 0) |
| param.s[1] | inside width (> 0) |
| param.s[2] | outer left coordinate |
| param.s[3] | outer top coordinate |

Creates an unmapped (not displayed) transparent window. Coordinates are relative to the inside of the parent. The tilemode of the new window is **TileModeRelative**. The clipmode of the new window is **ClipModeClipped**. The window does not have a defined cursor.

The reply:

**param.l[0]**        Window

Errors: **BadWindow**, **BadValue**


# X_DestroyWindow

**code**          3
**windowId**      Window

Unmaps and destroys the window and all of its subwindows. The windows should never again be referenced.

Windows are automatically destroyed when the creating process closes its network connection.

If the window has a mapped icon window, that icon window is unmapped and receives an **UnmapWindow** event. If the window is a mapped icon window, its corresponding regular window is mapped.

Generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**


# X_DestroySubwindows

**code**          4
**windowId**      Window

Destroy all subwindows of this window. The windows should never again be referenced.

Generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

# X_MapWindow

**code**         5
**windowId**      Window

Maps and raises the window and displays the window and all of its subwindows which have had map requests. The previous contents of all opaque windows are lost; mapping transparent windows does not affect the screen.

Mapping a window when one of its ancestors is unmapped does not cause the window to be displayed.

Has no effect if the window is already mapped.

Generates an **ExposeWindow** event on each newly displayed opaque window.

Errors: **BadWindow**


# X_MapSubwindows

**code**         6
**windowId**      Window

Maps all subwindows of the given window in an unspecified order.

Generates an **ExposeWindow** event on each newly displayed opaque window.

Errors: **BadWindow**


# X_UnmapWindow

**code**         7
**windowId**      Window

Unmaps the window and all of its subwindows. Unmapping transparent windows does not affect the screen.

Generates an **UnMapWindow** event on the window if it is mapped; generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

## X_UnmapSubwindows

**code**       8
**windowId**   Window

Unmaps all subwindows of the given window.

Generates an **UnMapWindow** event on each mapped subwindow; generates exposure events on formerly obscured opaque windows.


## X_UnmapTransparent

**code**       9
**windowId**   Window

Unmaps the window and all of its subwindows, but does not affect the screen, even if the window is opaque, and does not generate any exposure (or unmap) events.

Errors: **BadWindow**


## X_RaiseWindow

**code**       10
**windowId**   Window

Raise this window above all sibling windows, so that no sibling obscures it.

Raising a transparent window does not affect the screen. Transparent windows never obscure other windows for the purposes of output, but do obscure for the purposes of cursor and input control.

Window hierarchies never interleave. If window A is obscured by window B, then window A obscures only ancestors of B that are also ancestors of A.

Generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

# X_LowerWindow

| code | 11 |
| --- | --- |
| windowId | Window |

Lower this window below all sibling windows, so that it does not obscure any siblings.

Raising a transparent window does not affect the screen.

Generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

# X_CircWindowUp

| code | 12 |
| --- | --- |
| windowId | Window |

Raise the lowest mapped child of this window that is partially obscured by another child. Repeated executions lead to consecutive raising.

Generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

# X_MoveWindow

| code | 13 |
| --- | --- |
| func | 0 |
| windowId | Window |
| param.s[0] | outer left coordinate (start of border) |
| param.s[1] | outer top coordinate (start of border) |

Moves and raises the window, without changing its size. Coordinates are relative to the inside of the parent.

The contents of an opaque window are lost if its tilemode is relative or if the window is obscured by non-children. Moving a transparent window does not affect the screen.

Generates an **ExposeWindow** event on the (opaque) window if its contents are lost; generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

# X_ChangeWindow

| | |
|---|---|
| code | 14 |
| func | 0 |
| windowId | Window |
| param.s[0] | inside height (not including borders) (> 0) |
| param.s[1] | inside width (not including borders) (> 0) |

Changes the size of the window and raises it, without changing its upper left hand coordinate. The contents of an opaque window are lost; changing a transparent window does not affect the screen.

Generates an **ExposeWindow** event on the (opaque) window; generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**, **BadValue**

# X_ConfigureWindow

| | |
|---|---|
| code | 15 |
| func | 0 |
| windowId | Window |
| param.s[0] | inside height (not including borders) (> 0) |
| param.s[1] | inside width (not including borders) (> 0) |
| param.s[2] | outer left coordinate (start of border) |
| param.s[3] | outer top coordinate (start of border) |

Changes the size and placement of the window and raises it. The contents of an opaque window are lost; configuring a transparent window does not affect the screen.

Generates an **ExposeWindow** event on the (opaque) window; generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**, **BadValue**

# X_ChangeBackground

| | |
|---|---|
| code | 16 |
| windowId | Window |
| param.l[0] | background tile Pixmap or 0 |

Change the background tile of a window. If no background pixmap is specified, the background pixmap of the window's parent is used (on the root window the default background is restored).

Does not change the current contents of the window.

Can only be performed on an opaque window.

The pixmap can be freed immediately if no further explicit references to it are to be made.

Errors: **BadWindow**, **BadMatch**, **BadPixmap**, **BadTile**

## X_ChangeBorder

| | |
|---|---|
| **code** | 17 |
| **windowId** | Window |
| **param.l[0]** | border tile Pixmap |

Change the border tile of a window and repaint the border.

Can only be performed on an opaque window that has a border.

The pixmap can be freed immediately if no further explicit references to it are to be made.

Errors: **BadWindow**, **BadMatch**, **BadPixmap**, **BadTile**

## X_TileMode

| | |
|---|---|
| **code** | 18 |
| **func** | 0: TileModeAbsolute, 1: TileModeRelative |
| **windowId** | Window |

Sets the tilemode of the window. With **TileModeAbsolute** (the normal case for opaque windows), tiles are laid out with the upper left corner of the window as an effective origin. With **TileModeRelative** (the normal case for transparent windows), tiles are laid out with the upper left corner of the closest parent window with an absolute tilemode as an effective origin. The tilemode affects painting of the background for exposures and for **X_Clear**, as well as the **X_TileFill** and **X_DrawFilled** requests.

Does not change the current contents of the window.

Errors: **BadWindow**, **BadValue**

# X_ClipMode

| | |
|---|---|
| code | 19 |
| func | 0: ClipModeClipped, 1: ClipModeDrawThru |
| windowId | Window |

Sets the clipmode of the window. With **ClipModeClipped** (the normal case), future output to the window is obscured by subwindows. With **ClipModeDrawThru**, future output to the window ignores subwindows and draws into them. In draw-thru mode, the most useful display functions are **GXxor** and **GXinvert**, so that displaying again erases what was displayed. Draw-thru mode is useful for drawing window outlines when moving or resizing windows.

The root window starts with **ClipModeDrawThru**.

Errors: **BadWindow**, **BadValue**


# X_QueryWindow

| | |
|---|---|
| code | 20 |
| windowId | Window |

Get facts about the window.

The reply:

| | |
|---|---|
| param.s[0] | inside height (not including borders) |
| param.s[1] | inside width (not including borders) |
| param.s[2] | outer left (start of border) |
| param.s[3] | outer right (start of border) |
| param.s[4] | border width |
| param.b[10] | 0: IsUnmapped, 1: IsMapped, 2: IsInvisible (mapped but some ancestor is unmapped) |
| param.b[11] | 0: IsTransparent, 1: IsOpaque, 2: IsIcon |
| param.l[3] | icon Window or opaque Window or 0 |
| param.l[4] | event mask (**X_SelectInput**) |

If **windowId** is a transparent window, **param.b[11]** is **IsTransparent** and **param.l[3]** is **0**. If **windowId** is a normal opaque window, **param.b[11]** is **IsOpaque** and **param.l[3]** contains the window's icon window, if one has been defined with an **X_SetIconWindow** request. If **windowId** is an icon window, **param.b[11]** is **IsIcon** and **param.l[3]** contains the icon's corresponding regular window.

Errors: **BadWindow**

# X_StoreName

| | |
|---|---|
| **code** | 21 |
| **windowId** | Window |
| **param.s[0]** | length of name in characters ( > = 0) |

Assigns a name to a window.  This request must be followed by the characters of the window name, followed by 0 to 3 bytes to make the length a multiple of four.

The name is typically used by a window manager (to create named icons, for example).

Errors: **BadWindow**, **BadValue**


# X_FetchName

| | |
|---|---|
| **code** | 22 |
| **windowId** | Window |

Returns the name of a window.  The reply:

| | |
|---|---|
| **param.s[0]** | number of characters |

The reply is followed by the specified number of characters of name, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadWindow**


# X_SetIconWindow

| | |
|---|---|
| **code** | 23 |
| **windowId** | Window |
| **param.l[0]** | icon Window or 0 |

Sets or clears the icon window for a window.  The icon window must be a sibling of the specified window, both windows must be opaque, and neither can already be an icon window.

An icon window should be used when the client wants to control the contents of the icon form.  A window manager should create an icon window automatically if the client does not provide one.

Errors: **BadWindow**, **BadMatch**

# X_SetResizeHint

| | |
|---|---|
| **code** | 24 |
| **windowId** | Window |
| **param.s[0]** | base height ($>=0$) |
| **param.s[1]** | height increment ($>0$) |
| **param.s[2]** | base width ($>=0$) |
| **param.s[3]** | width increment ($>0$) |

Defines the shape of a window. The inside height of the window should be the base height plus some multiple of the height increment, and the inside width of the window should be the base width plus some multiple of the width increment. These parameters are hints, in that **X_ChangeWindow** and **X_ConfigureWindow** do not check conformance.

A newly created window has a base height and width of zero, and height and width increments of one.

The base height and width must be non-negative, and the height and width increments must be positive.

The resize hints are typically used by a window manager.

Errors: **BadWindow**, **BadValue**


# X_GetResizeHint

| | |
|---|---|
| **code** | 25 |
| **windowId** | Window |

Returns the resize parameters.

The reply:

| | |
|---|---|
| **param.s[0]** | minimum height |
| **param.s[1]** | height increment |
| **param.s[2]** | minimum width |
| **param.s[3]** | width increment |

Errors: **BadWindow**

## X_DefineCursor

| code | 26 |
|---|---|
| **windowId** | Window |
| **param.l[0]** | Cursor or 0 |

If a cursor is specified, it is used when the mouse is in the window. If no cursor is specified, the parent's cursor is used when the mouse is in the window.

On the root window, with no cursor specified, the default cursor is restored.

Errors: **BadWindow**, **BadCursor**


## X_SelectInput

| code | 27 |
|---|---|
| **windowId** | Window |
| **param.l[0]** | event mask |

Defines which input events the window is interested in. If a window is not interested in an event, it usually propagates up to the closest ancestor that is interested. The bits of the mask are:

```
KeyPressed        0x0001      keyboard key pressed
KeyReleased       0x0002      keyboard key released
ButtonPressed     0x0004      mouse button pressed
ButtonReleased    0x0008      mouse button released
EnterWindow       0x0010      mouse entering window
LeaveWindow       0x0020      mouse leaving window
MouseMoved        0x0040      mouse moves within window
ExposeWindow      0x0080      full window changed and/or exposed
ExposeRegion      0x0100      region of window exposed
ExposeCopy        0x0200      region exposed by X_CopyArea
RightDownMotion   0x0400      mouse moves with right button down
MiddleDownMotion  0x0800      mouse moves with middle button down
LeftDownMotion    0x1000      mouse moves with left button down
UnmapWindow       0x2000      window is unmapped
FocusChange       0x4000      keyboard focus changed
```

Selecting **ExposeRegion** also selects **ExposeWindow**.

Details of each kind of event are given later.

Overrides all previous selections on the same window by all clients.

If a window has both **ButtonPressed** and **ButtonReleased** selected, then a **ButtonPressed** event in that window automatically grabs the mouse until all buttons are released, with events sent to windows as described for **X_GrabMouse**.

Input selection on the root window should be reserved for a window manager.

Errors: **BadWindow**

## X_GrabMouse

| | |
|---|---|
| **code** | 28 |
| **windowId** | Window |
| **param.l[0]** | Cursor |
| **param.l[1]** | event mask |

All future mouse events go only to windows for which the grabbing client has issued **X_SelectInput** commands. (The event mask temporarily overrides any **X_SelectInput** on the window.) If the client has not issued an **X_SelectInput** command on the window where the event would normally be sent, then the event is sent to the specified window, provided the event is specified in the mask and is not **EnterWindow** or **LeaveWindow**. Only the bits **ButtonPressed**, **ButtonReleased**, **EnterWindow**, **LeaveWindow**, **MouseMoved**, **LeftDownMotion**, **MiddleDownMotion**, and **RightDownMotion** are useful in the mask. The specified cursor is used regardless of what window the mouse is in.

This request fails if someone else has already grabbed the mouse and has not released it; the request overrides any other grab in progress for this client.

If the request is successful there is a reply, but it contains no information.

Errors: **BadWindow**, **BadCursor**, **BadGrab**

## X_GrabButton

| | |
|---|---|
| **code** | 29 |
| **mask** | button mask |
| **windowId** | Window |
| **param.l[0]** | Cursor |
| **param.l[1]** | event mask |

The button mask must have exactly one of **LeftMask**, **MiddleMask**, or **RightMask** set, and may have some combination of **ControlMask**, **MetaMask**, **ShiftMask**, and **ShiftLockMask** set. If the specified button is pressed when exactly the specified keys are down, this and all future mouse events are grabbed until all buttons are released, with events sent to windows as described for **X_GrabMouse**. During the grab, the specified cursor is used regardless of what window the mouse is in.

This request fails if some other client has already grabbed the same button/key combination and has not released it.

If the request is successful there is a reply, but it contains no information.

Errors: **BadWindow**, **BadCursor**, **BadValue**, **BadGrab**

# X_QueryMouse

| | |
|---|---|
| **code** | 30 |
| **windowId** | Window |

Returns the current mouse coordinates and the state of various keys and buttons.

The reply:

| | |
|---|---|
| **param.l[0]** | Window |
| **param.s[2]** | x coordinate |
| **param.s[3]** | y coordinate |
| **param.s[4]** | key and button state |

The coordinates of the mouse relative to window are given, even if the mouse is outside the window. If the mouse is also in a mapped child window, the child is returned, otherwise the return window is 0.

The high bits of the state parameter are the same as for the event detail in keyboard and mouse events, and are defined further below in the discussion of input events.

Errors: **BadWindow**

# X_InterpretLocator

| | |
|---|---|
| **code** | 31 |
| **windowId** | Window |
| **param.l[0]** | Locator |

Interprets the coordinate with respect to the window.

The reply:

| | |
|---|---|
| **param.l[0]** | Window |
| **param.s[2]** | x coordinate |
| **param.s[3]** | y coordinate |

The coordinates of the locator relative to the window are given, even if the locator is outside the window. If the locator is also in a mapped child window, the child is returned, otherwise the return window is 0.

Errors: **BadWindow**

# X_WarpMouse

| | |
|---|---|
| **code** | **32** |
| **windowId** | destination Window |
| **param.s[0]** | destination x coordinate |
| **param.s[1]** | destination y coordinate |
| **param.l[1]** | source Window |
| **param.s[4]** | source height |
| **param.s[5]** | source width |
| **param.s[6]** | source left coordinate |
| **param.s[7]** | source top coordinate |

Move the mouse to the destination position relative to the origin of the destination window, but only if the mouse is currently in a visible portion of the specified region of the source window.

If the source height is zero, the current height of the source window minus the source top coordinate is used. If the source width is zero, the current width of the source window minus the source left coordinate is used.

Errors: **BadWindow**

# X_FocusKeyboard

| | |
|---|---|
| **code** | **33** |
| **func** | 0 |
| **windowId** | Window |

Generates exposure events on formerly obscured opaque windows.

Errors: **BadWindow**

# X_QueryTree

| | |
|---|---|
| **code** | **35** |
| **windowId** | Window |

Returns the parent and child windows of the specified window.

The reply:

**param.l[0]**        parent Window (or 0 if none)

param.l[1]          number of child windows

The reply is followed by the specified number of child Window IDs, each Window ID being 4 bytes long. The children are listed in current stacking order, from bottom-most (first) to top-most (last).

Errors: **BadWindow**

# X_Clear

code            40
windowId        Window

Clear the window and repaint it with the background. The tiling origin is controlled by the tilemode.

A transparent window inherits its parent's background for this operation.

Errors: **BadWindow**

# X_PixFill

code            41
func            display function (0-15)
mask            plane mask
windowId        Window
param.s[0]      destination height
param.s[1]      destination width
param.s[2]      destination left coord
param.s[3]      destination top coord
param.u[4]      source pixel
param.l[3]      mask Bitmap or 0

Performs a function in a region of the window. The source pixel defines the value of the source bit for each plane, and the plane mask defines which destination bit planes are affected. The display function is computed on each bit plane. If no mask bitmap is specified, the entire destination is affected. If a mask bitmap is specified, it defines the shape of the source and which pixels of the destination are affected.

Errors: **BadWindow**, **BadValue**, **BadBitmap**

# X_TileFill

| code | 42 |
|------|-----|
| func | display function (0-15) |
| mask | plane mask |
| windowId | Window |
| param.s[0] | destination height |
| param.s[1] | destination width |
| param.s[2] | destination left coord |
| param.s[3] | destination top coord |
| param.l[2] | tile Pixmap |
| param.l[3] | mask Bitmap or 0 |

Performs a function in a region of the window using a repeating pattern defined by the tile pixmap. The tiling origin is controlled by the window's tilemode. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane. If no mask bitmap is specified, the entire destination is affected. If a mask bitmap is specified, it defines which pixels of the destination are affected, and must be the same height and width as the destination.

Errors: **BadWindow**, **BadValue**, **BadPixmap**, **BadTile**, **BadBitmap**, **BadMatch**

# X_PixmapPut

| code | 43 |
|------|-----|
| func | display function (0-15) |
| mask | plane mask |
| windowId | Window |
| param.s[0] | source height |
| param.s[1] | source width |
| param.s[2] | source left coord |
| param.s[3] | source top coord |
| param.l[2] | source Pixmap |
| param.s[6] | destination left coord |
| param.s[7] | destination top coord |

Performs a function in a region of the window using a region of a pixmap. The source height, width, and coordinates specify the region of the source pixmap to be used. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane.

Errors: **BadWindow**, **BadValue**, **BadPixmap**

# X_PixmapBitsPut

| | |
|---|---|
| code | 44 |
| func | display function (0-15) |
| mask | plane mask |
| windowId | Window |
| param.s[0] | source height (> 0) |
| param.s[1] | source width (> 0) |
| param.s[2] | destination left coord |
| param.s[3] | destination top coord |
| param.s[4] | 0: XYFormat, 1: ZFormat |
| param.l[3] | mask Bitmap or 0 |

Performs a function in a region of the window using a pixmap. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane. If a mask bitmap is specified, it defines which pixels of the destination are affected, and must be the same height and width as the source.

The request must be followed by the data bytes of the source pixmap in the specified format, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadWindow**, **BadValue**, **BadBitmap**, **BadMatch**

# X_BitmapBitsPut

| | |
|---|---|
| code | 45 |
| func | display function (0-15) |
| mask | plane mask |
| windowId | Window |
| param.s[0] | source height (> 0) |
| param.s[1] | source width (> 0) |
| param.s[2] | destination left coord |
| param.s[3] | destination top coord |
| param.u[4] | foreground pixel |
| param.u[5] | background pixel |
| param.l[3] | mask Bitmap or 0 |

Performs a function in a region of the window using a pixmap defined by a bitmap and a pair of source pixels. The foreground pixel defines the source for the one bits in the bitmap, and the background pixel defines the source for the zero bits. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane. If a mask bitmap is specified, it defines which pixels of the destination are affected, and must be the same height and width as the source.

The request must be followed by the data bytes of the source bitmap in bitmap format, followed by 0 or 2 pad bytes to make the length a multiple of four.

Errors: **BadWindow, BadValue, BadBitmap, BadMatch**

# X_CopyArea

| | |
|---|---|
| **code** | 46 |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | source height |
| **param.s[1]** | source width |
| **param.s[2]** | source left coord |
| **param.s[3]** | source top coord |
| **param.s[6]** | destination left coord |
| **param.s[7]** | destination top coord |

Copies one region of the window to another region in the same window. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane.

If parts of the source are obscured, the corresponding parts of the destination are filled with the window's background. If **ExposeCopy** has been selected, **ExposeRegion** events are generated for those parts of the destination, and then an **ExposeCopy** event is generated. All of these events are together in the stream, with no intervening events.

Errors: **BadWindow, BadValue**

# X_Text

| | |
|---|---|
| **code** | 47 |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | destination left coord |
| **param.s[1]** | destination top coord |
| **param.l[1]** | Font |
| **param.u[4]** | foreground pixel |
| **param.u[5]** | background pixel |
| **param.s[6]** | number of characters ( > = 0) |
| **param.b[14]** | inter-character pad |
| **param.b[15]** | space character pad |

Draws text using the specified function. The coordinates are for the upper left of the first character. The foreground pixel defines the source for the one bits in the font character bitmaps, and the background pixel defines the source for the zero bits. The plane mask

defines which destination bit planes are affected. The display function is computed on each bit plane. The inter-character pad specifies the number of pixels to skip after each character before printing the next character. The space character pad specifies the number of additional pixels to skip after each space character before printing the next character. The skipped pixels are not considered part of the source or destination, and are not altered.

The request must be followed by the specified number of characters, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadWindow**, **BadValue**, **BadFont**

## X_TextMask

| | |
|---|---|
| **code** | 48 |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | destination left coord |
| **param.s[1]** | destination top coord |
| **param.l[1]** | Font |
| **param.u[4]** | source pixel |
| **param.s[6]** | number of characters ( > = 0) |
| **param.b[14]** | inter-character pad |
| **param.b[15]** | space character pad |

Like **X_Text**, but the source pixel defines the value of the source bit for each plane, and the font character bitmaps are used as masks to define which pixels of the destination are affected.

The request must be followed by the specified number of characters, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadWindow**, **BadValue**, **BadFont**

## X_Line

| | |
|---|---|
| **code** | 49 |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | x1 coord |
| **param.s[1]** | y1 coord |
| **param.s[2]** | x2 coord |
| **param.s[3]** | y2 coord |

| | |
|---|---|
| **param.u[4]** | source pixel |
| **param.b[10]** | brush height (> 0) |
| **param.b[11]** | brush width (> 0) |

This request is the same as a DrawSolidLine X_Draw request with the two vertexes (x1, y1, VertexDontDraw) and (x2, y2, VertexDrawLastPoint).

Errors: **BadWindow**, **BadValue**


# X_Draw

| | |
|---|---|
| **code** | 50 |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | number of vertexes (> = 0) |
| **param.u[1]** | source pixel |
| **param.b[4]** | brush height (> 0) |
| **param.b[5]** | brush width (> 0) |
| **param.s[3]** | 0: DrawSolidLine, 1: DrawDashedLine, 2: DrawPatternedLine |
| **param.u[4]** | alternate source pixel (for patterned line) |
| **param.s[5]** | pattern string (for dashed or patterned line) |
| **param.s[6]** | pattern length (for dashed or patterned line) (1-16) |
| **param.s[7]** | pattern multiplier (for dashed or patterned line) (> 0) |

Draws arbitrary polygons/curves using the specified function and brush rectangle. The area covered should be that obtained by laying down the brush rectangle at every point along the path, with the upper left corner following the path. Each pixel in that area may be processed just once, or the brush may be repainted for each point along the path. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane. For a solid line, the source pixel defines the value of the source bit for each plane. For a dashed or patterned line, the pattern string specifies up to 16 bits of pattern; the pattern length specifies the number of bits. The pattern multiplier specifies the number of times each bit in the string should be repeated before moving to the next bit. The bits are used least significant first, wrapping as needed. For a dashed line, the destination is only updated when the pattern bit is 1. For a patterned line, the alternate source pixel is used when the pattern bit is 0.

The request must be followed by the specified number of vertexes, followed by 0 or 2 pad bytes to make the length a multiple of four. The C definition of a vertex is:

```
typedef struct _Vertex {
    short x, y;
    unsigned short flags;
} Vertex;
```

The flags are as follows:

| VertexRelative | 0x0001 else absolute |
| VertexDontDraw | 0x0002 else draw |
| VertexCurved | 0x0004 else straight |
| VertexStartClosed | 0x0008 else not |
| VertexEndClosed | 0x0010 else not |
| VertexDrawLastPoint | 0x0020 else don't |

A relative vertex is expressed in terms of offsets from the previous vertex, an absolute vertex has offsets from the origin of the window. The first vertex is never relative. **VertexDontDraw** and **VertexCurved** control drawing from the previous vertex to the current vertex. It can be useful to combine **VertexDontDraw** and **VertexCurved** to define the shape of the displayed portion of the curve. **VertexDontDraw** can also be used to combine multiple draws in one request. **VertexStartClosed** should be set in the first vertex of a closed curve, **VertexEndClosed** in the last; the two should specify the same point. In drawing from the previous vertex to the current vertex, the current vertex point is not drawn unless **VertexDrawLastPoint** is set.

Errors: **BadWindow**, **BadValue**

# X_DrawFilled

| **code** | **51** |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | number of vertexes ($>=0$) |
| **param.u[1]** | source pixel |
| **param.l[1]** | tile Pixmap or 0 |

Draws arbitrary filled polygons/curves using the specified function tiling pixmap or source pixel. If a tile is specified, the tiling origin is controlled by the window's tilemode. If no tile is given, the source pixel is used instead. The plane mask defines which destination planes are affected.

The request must be followed by the specified number of vertexes, followed by 0 or 2 pad bytes to make the length a multiple of four.

The vertex list should only consist of one or more closed regions. A point is defined to be inside a region if an infinite ray with the point as an origin crosses the path of the region an odd number of times.

Errors: **BadWindow**, **BadValue**, **BadPixmap**, **BadTile**

# X_PixmapSave

| | |
|---|---|
| **code** | **52** |
| **windowId** | Window |
| **param.s[0]** | height (> 0) |
| **param.s[1]** | width (> 0) |
| **param.s[2]** | left coord |
| **param.s[3]** | top coord |

Creates a pixmap from the given portion of the window. The pixmap contains a direct image of that portion of the screen, including any visible portions of subwindows or overlapping windows.

The window must be mapped, and it must be the case that, if there were no subwindows or overlapping windows, the specified portion of the window would be fully visible on the screen.

The reply:

| | |
|---|---|
| **param.l[0]** | Pixmap |

Errors: **BadWindow, BadValue, BadAlloc**

# X_PixmapGet

| | |
|---|---|
| **code** | **53** |
| **func** | 0: XYFormat, 1: ZFormat |
| **windowId** | Window |
| **param.s[0]** | height (> 0) |
| **param.s[1]** | width (> 0) |
| **param.s[2]** | left coord |
| **param.s[3]** | top coord |

Returns the contents of the given portion of the window in the given pixmap format. The pixmap contains a direct image of that portion of the screen, including any visible portions of subwindows or overlapping windows.

The window must be mapped, and it must be the case that, if there were no subwindows or overlapping windows, the specified portion of the window would be fully visible on the screen.

The reply:

| | |
|---|---|
| **param.l[0]** | number of bytes |

The reply is followed by the specified number of bytes of pixmap data, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadWindow, BadValue**

# X_StippleFill

| | |
|---|---|
| **code** | **54** |
| **func** | display function (0-15) |
| **mask** | plane mask |
| **windowId** | Window |
| **param.s[0]** | destination height |
| **param.s[1]** | destination width |
| **param.s[2]** | destination left coord |
| **param.s[3]** | destination top coord |
| **param.u[4]** | source pixel |
| **param.l[3]** | stipple Bitmap |

Performs a function a a region of the window using a repeating pattern defined by the stipple bitmap. The tiling origin is controlled by the window's tilemode. The plane mask defines which destination bit planes are affected. The display function is computed on each bit plane. The destination pixels written are those corresponding to the 1 values in the pattern generated from the stipple bitmap.

Errors: **BadWindow**, **BadValue**, **BadTile**, **BadBitmap**

# X_SetUp

| | |
|---|---|
| **code** | **80** |

Returns the root window and device information.

The reply:

| | |
|---|---|
| **param.l[0]** | root Window |
| **param.s[2]** | protocol version number (10) |
| **param.s[3]** | device identifier |
| **param.s[4]** | number of bit planes |
| **param.u[5]** | number of usable color map cells |

IBM device identifiers range from 200 to 299. IBM device identifiers are constructed as follows:

**200 + display_device_identifier + keyboard_device_identifier**

The folowing IBM display and keyboard device identifiers are currently defined:

| | | |
|---|---|---|
| XDEV_IBM_BASE | 200 | base for IBM devices |
| XDEV_IBM_6153 | 10 | advanced mono display |
| XDEV_IBM_6154 | 20 | advanced color display |
| XDEV_IBM_6155 | 30 | extended mono display |
| XDEV_IBM_MEGA | 40 | extended color display |
| XDEV_IBM_K101 | 0 | US keyboard (101 keys) |
| XDEV_IBM_K102 | 1 | European keyboard (102 keys) |
| XDEV_IBM_K106 | 2 | Japanese keyboard (106 keys) |

See **/usr/include/X/Xproto.h** for a list of the currently known devices.

## X_UngrabMouse

**code**            81

Releases hold of the mouse if it was grabbed with an **X_GrabMouse**.

## X_UngrabButton

**code**            82
**mask**            button mask

Releases hold of the button/key combination if it was grabbed. The button mask must have exactly one of **LeftMask**, **MiddleMask**, or **RightMask** set, and may have some combination of **ControlMask**, **MetaMask**, **ShiftMask**, and **ShiftLockMask** set.

Errors: **BadValue**

## X_GetColor

**code**            83
**param.u[0]**       red value
**param.u[1]**       green value
**param.u[2]**       blue value

Determines the closest color provided by the hardware, and returns a pixel value representing that color. The corresponding color map cell is read-only.

Read-only color map cells are shared among clients, so this request may simply reference count an existing cell.

The reply:

**param.u[0]**          pixel

Errors: **BadAlloc**

# X_GetColorCells

| | |
|---|---|
| **code** | 84 |
| **func** | 1 if planes must be contiguous, 0 otherwise |
| **param.s[0]** | number of colors ( > = 0) |
| **param.s[1]** | number of planes ( > = 0) |

Allocates $n*2^P$ color map cells, where **n** is the number of colors and P is the number of planes specified.

The reply:

**param.u[0]**          plane mask

The bits in the plane mask are contiguous if requested. The reply is followed by 2 * N bytes of data, where N is the number of colors specified, followed by 0 or 2 pad bytes to make the length a multiple of four. Each 16-bit word represents the pixel value of one of the color cells. Additional pixel values are obtained by *oring* in all possible combinations of one bits from the plane mask. The initial colors for all of these cells is undefined.

If zero colors are requested, then the request allocates all cells with a pixel value having at least one non-zero bit in the plane mask. At most one such request succeeds.

Errors: **BadValue**, **BadAlloc**

# X_FreeColors

| | |
|---|---|
| **code** | 85 |
| **mask** | plane mask |
| **param.s[0]** | number of colors ( > = 0) |

Frees several colors or color map cells. Further use of the given pixel values results in undefined colors.

The request must be followed by the specified number of 16-bit pixel values, followed by 0 or 2 pad bytes to make the length a multiple of four. Additional pixel values are obtained by oring in all possible combinations of one bits from the plane mask.

Errors: **BadValue**, **BadAccess**

## X_StoreColors

| | |
|---|---|
| **code** | 86 |
| **param.s[0]** | number of colors ( > = 0) |

Change the colors of several pixels to the closest available hardware colors.

The request must be followed by the specified number of color definitions.  The C definition is:

```
typedef struct _ColorDef {
    unsigned short pixel;
    unsigned short red, green, blue;
} ColorDef;
```

Errors: **BadValue, BadAccess**

## X_QueryColor

| | |
|---|---|
| **code** | 87 |
| **param.u[0]** | pixel |

Returns the color values for a pixel.

The reply:

| | |
|---|---|
| **param.u[0]** | red value |
| **param.u[1]** | green value |
| **param.u[2]** | blue value |

Errors: **BadValue**

## X_GetFont

| | |
|---|---|
| **code** | 88 |
| **param.s[0]** | length of name in chars ( > 0) |

Loads a font.  This request must be followed by the characters of the font name, followed by 0 to 3 pad bytes to make the length a multiple of four.  Case is significant.

Fonts are shared among clients, so this request may simply reference count an existing font.

The reply:

| | |
|---|---|
| **param.l[0]** | Font |

Errors: **BadValue, BadFont, BadAlloc**

## X_FreeFont

**code**              89
**param.l[0]**      Font

Indicates that the font is no longer needed.  The font should never again be referenced.

Errors: **BadFont**


## X_QueryFont

**code**              90
**param.l[0]**      Font

Returns information about a font.

The reply:

**param.s[0]**      height
**param.s[1]**      average width
**param.s[2]**      first character
**param.s[3]**      last character
**param.s[4]**      baseline
**param.s[5]**      1 if fixed width, 0 if variable width

The baseline specifies where in pixels from the bottom of the font the characters without descenders begin.

A font is fixed width if all characters in the given range are the same width.

Errors: **BadFont**


## X_CharWidths

**code**              91
**param.l[0]**      Font
**param.s[2]**      number of characters ( $> = 0$ )

Returns the width in pixels of each character.  The request must be followed by the specified number of characters, followed by 0 to 3 pad bytes to make the length a multiple of four.

The reply:

**param.l[0]**      number of bytes

The reply is followed by the specified number of bytes of data, followed by 0 or 2 pad bytes to make the length a multiple of four. Each 16-bit word of data contains the width of a character.

Errors: **BadFont**, **BadValue**

# X_StringWidth

| | |
|---|---|
| **code** | 92 |
| **param.l[0]** | Font |
| **param.s[2]** | number of characters ( > = 0) |

Returns the width in pixels of a string in a font.

The request must be followed by the specified number of characters, followed by 0 to 3 pad bytes to make the length a multiple of four.

The reply:

| | |
|---|---|
| **param.s[0]** | width in pixels |

Errors: **BadFont**, **BadValue**

# X_FontWidths

| | |
|---|---|
| **code** | 93 |
| **param.l[0]** | Font |

Returns the widths in pixels of all characters in a font.

The reply:

| | |
|---|---|
| **param.l[0]** | number of bytes |

The reply is followed by the specified number of bytes of data, followed by 0 or 2 pad bytes to make the length a multiple of four. Each 16-bit word of data contains the width of a character. The widths are for the range of characters given by **X_QueryFont**.

Errors: **BadFont**

## X_StoreBitmap

| | |
|---|---|
| **code** | **94** |
| **param.s[0]** | height ( > 0) |
| **param.s[1]** | width ( > 0) |

Creates a bitmap.

The request must be followed by the correct number of bytes of data in bitmap format, followed by 0 or 2 pad bytes to make the length a multiple of four.

The reply:

| | |
|---|---|
| **param.l[0]** | Bitmap |

Errors: **BadValue, BadAlloc**

## X_FreeBitmap

| | |
|---|---|
| **code** | **95** |
| **param.l[0]** | Bitmap |

Frees the storage consumed by the bitmap. The bitmap should never be referenced again.

Errors: **BadBitmap**

## X_CharBitmap

| | |
|---|---|
| **code** | **96** |
| **param.l[0]** | Font |
| **param.s[2]** | character |

Copies a character bitmap from a font.

The font can be freed immediately if no further explicit references to it are to be made.

The reply:

| | |
|---|---|
| **param.l[0]** | Bitmap |

Errors: **BadFont, BadValue, BadAlloc**

# X_StorePixmap

| | |
|---|---|
| **code** | 97 |
| **func** | 0: XYFormat, 1: ZFormat |
| **param.s[0]** | height (> 0) |
| **param.s[1]** | width (> 0) |

Creates a pixmap.

The request must be followed by the correct number of bytes of data the specified format, followed by 0 to 3 pad bytes to make the length a multiple of four.

The reply:

| | |
|---|---|
| **param.l[0]** | Pixmap |

Errors: **BadValue, BadAlloc**


# X_FreePixmap

| | |
|---|---|
| **code** | 98 |
| **param.l[0]** | Pixmap |

Frees the storage consumed by the pixmap. The pixmap should never be referenced again.

Errors: **BadPixmap**


# X_MakePixmap

| | |
|---|---|
| **code** | 99 |
| **param.l[0]** | Bitmap or 0 |
| **param.u[2]** | foreground pixel |
| **param.u[3]** | background pixel |

Creates a pixmap from a bitmap. The foreground pixel is used for the one bits in the bitmap, and the background pixel is used for the zero bits. If no bitmap is given, a bitmap of all one bits suitable for use as a tiling pixmap is used.

The bitmap can be freed immediately if no further explicit references to it are to be made.

The reply:

| | |
|---|---|
| **param.l[0]** | Pixmap |

Errors: **BadBitmap, BadValue, BadAlloc**

# X_QueryShape

| | |
|---|---|
| **code** | **100** |
| **func** | 0: CursorShape, 1: TileShape, 2: BrushShape |
| **param.s[0]** | height (> 0) |
| **param.s[1]** | width (> 0) |

Given a rectangular shape, returns the closest shape actually supported by the display for a given purpose. For a cursor shape, returns a Bitmap shape acceptable for **X_StoreCursor**. For a tile shape, returns a Pixmap shape acceptable for tiling. For a brush shape, returns a shape acceptable for **X_Line** and **X_Draw**.

The reply:

| | |
|---|---|
| **param.s[0]** | height |
| **param.s[1]** | width |

Errors: **BadValue**


# X_StoreCursor

| | |
|---|---|
| **code** | **101** |
| **func** | display function (0-15) |
| **param.l[0]** | cursor Bitmap |
| **param.u[2]** | foreground pixel |
| **param.u[3]** | background pixel |
| **param.l[2]** | mask Bitmap or 0 |
| **param.s[6]** | x offset |
| **param.s[7]** | y offset |

Defines a mouse cursor. The foreground pixel is used for the one bits in the cursor bitmap, and the background pixel is used for the zero bits. The mask bitmap defines the shape of the cursor; that is, the one bits in the mask define which cursor pixels are displayed. If no mask is given, all pixels of the cursor are displayed. The mask bitmap, if present, must be the same size as the cursor bitmap. The offsets define the point that actually corresponds to the mouse position; this must be a point in the cursor bitmap.

The components of the cursor may be transformed arbitrarily to meet hardware limitations.

The bitmaps can be freed immediately if no further explicit references to them are to be made.

The reply:

| | |
|---|---|
| **param.l[0]** | Cursor |

Errors: **BadValue**, **BadBitmap**, **BadMatch**, **BadAlloc**

## X_FreeCursor

| | |
|---|---|
| **code** | 102 |
| **param.l[0]** | Cursor |

Frees the storage consumed by the cursor. The cursor should never be referenced again.

Errors: **BadCursor**


## X_MouseControl

| | |
|---|---|
| **code** | 103 |
| **param.s[0]** | acceleration ( > = 1) |
| **param.s[1]** | threshold ( > = 0) |

Defines how the mouse moves. The acceleration is a multiplier for movement. For example, specifying 3 means the cursor moves three times as fast as the mouse. Acceleration only takes effect if the mouse moves more than threshold pixels at once, and only applies to the amount beyond the threshold.

Errors: **BadValue**


## X_FeepControl

| | |
|---|---|
| **code** | 104 |
| **func** | volume (0-7) |

Defines the base volume for **X_Feep** requests. The volume is in the range 0 to 7, with 7 the loudest.

Errors: **BadValue**


## X_Feep

| | |
|---|---|
| **code** | 105 |
| **param.s[0]** | relative volume (-7 to 7) |

Causes an audible bell. The volume is added to the base volume defined by the **X_FeepControl** request, the sum limited to the range 0 to 7.

Errors: **BadValue**

# X_ShiftLock

| | |
|---|---|
| **code** | **106** |
| **func** | 0: LockUpDownMode, 1: LockToggleMode |

Sets the mode of the Shift LOCK key on the keyboard. When the keyboard is in **LockUpDownMode**, **KeyPressed** and **KeyReleased** events are sent as for any other key, and the **ShiftLockMask** sent in events gives the current state of the key. In **LockToggleMode**, **KeyPressed** and **KeyReleased** events are never sent for the LOCK key, and the state of the **ShiftLockMask** sent in events is toggled on every press of the LOCK key.

The key is initially in **LockToggleMode**.

Errors: **BadValue**

# X_KeyClick

| | |
|---|---|
| **code** | **107** |
| **func** | volume (0-8) |

Turns keyboard key click off (volume 0), or turns it on and sets the volume, with 8 the loudest.

Errors: **BadValue**

# X_AutoRepeat

| | |
|---|---|
| **code** | **108** |
| **func** | 0 for off, 1 for on |

Turns keyboard autorepeat on or off.

Errors: **BadValue**

# X_ScreenSaver

| | |
|---|---|
| **code** | **109** |
| **func** | 0 for video off, 1 for video on |
| **param.s[0]** | screen saver timeout in minutes (> 0) |
| **param.s[1]** | pattern change timeout interval in minutes (> 0) |

If the server remains idle for the specified number of minutes, screen saver is enabled. If video off is specified, and the hardware supports video blanking, the screen goes blank.

Otherwise, the screen is tiled with the root window background tile, randomly re-origined at the specified timeout interval. The screen state is restored when the next request or input event occurs.

Errors: **BadValue**


# X_StoreBytes

| | |
|---|---|
| **code** | **110** |
| **func** | cut buffer (0-7) |
| **param.s[0]** | number of bytes ( > = 0) |

Stores an arbitrary string of bytes one of eight cut buffers. These bytes may be retrieved with the **X_FetchBytes** request.

The previous contents of the cut buffer are lost.

The request must be followed by the specified number of bytes of data, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadValue**


# X_FetchBytes

| | |
|---|---|
| **code** | **111** |
| **func** | cut buffer (0-7) |

Retrieves the contents of the specified cut buffer.

The reply:

| | |
|---|---|
| **param.s[0]** | number of bytes |

The reply is followed by the specified number of bytes of data, followed by 0 to 3 pad bytes to make the length a multiple of four.

Errors: **BadValue**

## X_RotateCuts

| | |
|---|---|
| **code** | 112 |
| **func** | rotate count (0-7) |

Rotates the cut buffers by N.  Buffer 0 becomes buffer N, buffer 1 becomes N + 1 mod 8, and so on.

Errors: **BadValue**

## X_AddHost

| | |
|---|---|
| **code** | 113 |
| **func** | address family |
| **param.l[0-?]** | host address |

Add the specified host to the list of hosts allowed to open connections.

The currently recognized address families are:

**XAF_INET**    2

In AIX terms, the host address for **XAF_INET** is **struct in_addr**.

The client must reside on the same host as the window system.

Under AIX, the initial set of hosts consists of the host the window system is running on, plus those hosts listed in **/etc/X?.hosts**, where *?* is the number of the display.  This file should consist of host names separated by new lines.

Errors: **BadValue, BadAccess**

## X_RemoveHost

| | |
|---|---|
| **code** | 114 |
| **func** | address family |
| **param.l[0-?]** | host address |

Remove the specified host from the list of hosts allowed to open connections.

The address family and host address are as for **X_AddHost**.

The client must reside on the same host as the window system.

Errors: **BadValue, BadAccess**

# X_GetHosts

| | |
|---|---|
| **code** | **115** |
| **func** | address family |

Returns the current list of hosts allowed to open connections.

The reply:

| | |
|---|---|
| **param.l[0]** | number of bytes |

The reply is followed by the specified number of bytes of data, containing a list of host addresses. The size of each host address is determined by the address family, as for **X_Addhost**.

Errors: **BadValue**

# X_GrabServer

| | |
|---|---|
| **code** | **116** |

Disables processing of requests and close-downs on all other connections (than the one this request arrived on).

# X_UngrabServer

| | |
|---|---|
| **code** | **117** |

Restarts processing of requests and close-downs on other connections.

# X_LookupColor

| | |
|---|---|
| **code** | **118** |
| **param.s[0]** | length of name in characters ( > = 0) |

Returns the color values for a given color name. The name is looked up in a standard color database. This request must be followed by the characters of the color name, followed by 0 to 3 pad bytes to make the length a multiple of four. Case is significant.

The reply:

| | |
|---|---|
| **param.u[0]** | exact red value |
| **param.u[1]** | exact green value |
| **param.u[2]** | exact blue value |
| **param.u[3]** | closest hardware red value |

| param.u[4] | closest hardware green value |
| param.u[5] | closest hardware blue value |

Errors: **BadValue**, **BadColor**

# Input Events

Selecting **MouseMoved** results in events independent of the state of the mouse buttons. By selecting some subset of (**LeftDownMotion**, **MiddleDownMotion**, **RightDownMotion**) instead, **MouseMoved** events are only generated when one or more of the specified buttons is depressed.

**KeyPressed**, **KeyReleased**, **ButtonPressed**, **ButtonReleased**, and **MouseMoved** events are usually sent to the smallest window enclosing the mouse that has selected such events. For **KeyPressed** and **KeyReleased** events, if this window is not in the keyboard focus hierarchy, the events are sent to the focus window instead.

**KeyPressed**, **KeyReleased**, **ButtonPressed**, **ButtonReleased**, **EnterWindow**, **LeaveWindow**, and **MouseMoved** events have the following structure:

| code | kind of event (**KeyPressed**, etc.) |
| param.l[0] | event Window |
| param.s[2] | time in 10 millisecond ticks (Button only) |
| param.s[3] | event detail |
| param.s[4] | mouse x coord within event window |
| param.s[5] | mouse y coord within event window |
| param.l[3] | sub Window |
| param.l[4] | Locator |

The coordinates of the mouse relative to the event window are reported, even if the mouse is not in the window (because of grabbing or keyboard focusing). If the mouse is also in a (direct) child of the event window, the subwindow is set to that child, otherwise the subwindow is 0. The locator defines the mouse coordinates in absolute terms.

The time value is present only for **ButtonPressed** and **ButtonReleased** events. Note that there are only 16 bits of time (which should be treated as unsigned), which wraps after approximately 11 minutes, so only time differences between clustered events are interesting.

For all seven event types, the high bits of the detail encode the state of various keys and buttons just before the event:

```
ControlMask          0x4000 Control key
MetaMask             0x2000 Meta (Symbol) key
ShiftMask            0x1000 Shift key
ShiftLockMask        0x0800 ShiftLock key
LeftMask             0x0400 Left button
MiddleMask           0x0200 Both buttons
RightMask            0x0100 Right button
```

For **KeyPressed** and **KeyReleased**, the low byte of the detail gives the key. This is not an ASCII character, but a keyboard scan code.

For **ButtonPressed** and **ButtonReleased**, the low byte of the detail is one of:

```
0                    RightButton
1                    MiddleButton
2                    LeftButton
```

For **EnterWindow** and **LeaveWindow**, the low byte of the detail is either zero or one of:

```
1                    IntoOrFromSubwindow
2                    VirtualCrossing
```

**EnterWindow** and **LeaveWindow** events are generated as follows:

- When the mouse moves from window A to window B, and B is an ancestor of A:

  - A gets a **LeaveWindow** with detail 0
  - Windows between A and B exclusive that have **LeaveWindow** selected get a **LeaveWindow** with detail 2
  - B gets an **EnterWindow** with detail 1

- When the mouse moves from window A to window B, and B is a descendant of A:

  - A gets a **LeaveWindow** with detail 1
  - Windows between A and B exclusive that have **EnterWindow** selected get an **EnterWindow** with detail 2
  - B gets an **EnterWindow** with detail 0

- When the mouse moves from window A to window B, with window C being their least common ancestor:

  - A gets a **LeaveWindow** with detail 0
  - Windows between A and C exclusive that have **LeaveWindow** selected get a **LeaveWindow** with detail 2
  - Windows between C and B exclusive that have **EnterWindow** selected get an **EnterWindow** with detail 2
  - B gets an **EnterWindow** with detail 0

- At the start of a mouse grab, either automatically from a button press, or from an **X_GrabMouse** or **X_GrabButton**, with the mouse in window A, and with window B being the smallest window enclosing the mouse that has had an **X_SelectInput** issued on it by some client:

    - A gets a **LeaveWindow** with detail 0 if the grabbing client has not issued an **X_SelectInput** command on B

    - Ancestors of A (not including the root) get a **LeaveWindow** with detail 2 if the grabbing client has not issued an **X_SelectInput** on the window and the window has **LeaveWindow** selected.

- At the end of a mouse grab, with the mouse in window A, and with window B being the smallest window enclosing the mouse that has had an **X_SelectInput** issued on it by some client:

    - Ancestors of A (not including the root) get an **EnterWindow** with detail 2 if the grabbing client has not issued an **X_SelectInput** on the window and the window has **EnterWindow** selected.

    - A gets an **EnterWindow** with detail 0 if the grabbing client has not issued an **X_SelectInput** command on B.

**Note:** **EnterWindow** and **LeaveWindow** events with detail 0 or 1 (but not 2) propagate to the smallest enclosing window that has actually selected the event.

**LeaveWindow** events are not generated when windows are unmapped or destroyed.

**UnmapWindow** events occur whenever an **X_UnmapWindow** or **X_UnmapSubwindows** request is executed on a mapped window. The event structure is:

| | |
|---|---|
| **code** | **UnmapWindow** |
| **param.l[0]** | event Window |
| **param.l[3]** | sub Window |

If a subwindow is given, it is the actual window on which the request was issued (not the ancestor that is a direct child of the event window).

**FocusChange** events occur whenever the keyboard focus changes. The event structure is:

| | |
|---|---|
| **code** | **FocusChange** |
| **param.l[0]** | event Window |
| **param.s[3]** | **EnterWindow** or **LeaveWindow** |
| **param.l[3]** | sub Window |

If a subwindow is given, it is the actual window on which the request was issued (not the ancestor that is a direct child of the event window).

For **ExposeWindow** and **ExposeRegion** events, the structure is as follows:

| | |
|---|---|
| **code** | **ExposeWindow** or **ExposeRegion** |
| **param.l[0]** | event Window |
| **param.s[3]** | detail (0 or ExposeCopy) |

| | |
|---|---|
| **param.s[4]** | width of area |
| **param.s[5]** | height of area |
| **param.l[3]** | sub Window |
| **param.s[8]** | top coord of area |
| **param.s[9]** | left coord of area |

Coordinates are relative to the inside of the exposed window.

**ExposeWindow** and **ExposeRegion** events are triggered as (parts of) windows become exposed. When an entire window becomes exposed (as when a window is mapped or changes size), an **ExposeWindow** event is sent. The width and height of the entire window is given, and the coordinates are (0, 0). When only parts of a window become exposed (as when an obscuring window is moved), **ExposeRegion** events are sent describing each newly exposed area. However, if only ExposeWindow has been selected, a single **ExposeWindow** is sent instead. If the region exposure is the result of a **CopyArea**, **ExposeCopy** is set in the detail word. If the exposure is actually that of a descendant of the window selecting the event, the subwindow is set to that descendant and the coordinates are actually for the subwindow, otherwise the subwindow is 0. For a given window exposure or **CopyArea**, all resulting **ExposeRegion** events are sent contiguously, with no other events interspersed.

For **ExposeCopy** events, the structure is as follows:

**code**              **ExposeCopy**
**param.l[0]**        event Window
**param.l[3]**        sub Window

If the **CopyArea** was done in a descendant of the window selecting the event, the subwindow is set to that descendant, otherwise the subwindow is 0.

# Appendix A.  X-Windows Installation

This appendix explains how to install the X-Windows licensed program.  The following tasks must be performed before you install X-Windows.

- Install the AIX Operating System.  See *Installing and Customizing the AIX Operating System*.

- Install the Advanced Display Graphics Support Library from Multi-User Services.

Before you can use X-Windows, you must create a number of pty devices using the **devices** command.  Add a pty device for each **xterm** window you plan to use.  The procedure to add pty devices is contained in *Installing and Customizing the AIX Operating System*.

Make sure that no one else is using the system and that no user programs are running before you install the X-Windows licensed program.  If other users are working on the system, installation may fail.

## Operating from AIX Shell or Usability Services

You must be in AIX Shell or Usability Services to install the X-Windows licensed program.  If you are now using AIX Shell, go to "Installing X-Windows from the AIX Shell" on page A-2.

If you are using Usability Services, you have two choices:

- Turn to the customization **install** and **devices** commands, described in *Usability Services Reference*.  You can select the **install** and **devices** commands and follow the prompts.

- Go to the WINDOWS window and select AIX from the Window Types pane.  Select OPEN from the command bar.  Enter the **installp** and the **devices** commands and follow the prompts.

# Installing X-Windows from the AIX Shell

To install the X-Windows licensed program from the AIX Shell, follow the steps on this page. If an error message occurs during the procedure, see *Messages Reference* for details.

---

**To Install X-Windows**

1. Log in as superuser (su) or / (root).

2. Add pty devices using the **devices** command.

3. Make sure no one else is using the system and that no user programs are running.

4. Type             . Follow the prompts to insert the X-Windows licensed program diskettes and install the program.

---

## More Detailed Information

1. Log on the system as superuser (su) or / (root). After logging in, you see the AIX Operating System # prompt.

```
IBM RT PC AIX Operating System
(C) COPYRIGHT IBM CORP. 1985, 1987
(/dev/console)
login: su


#
```

See *Using the AIX Operating System* if you require more information.

2. Run the **devices** command. For step-by-step information about running devices, see *Installing and Customizing the AIX Operating System*. For more information about the **devices** command, see *AIX Operating System Commands Reference*.

   **Note:** For **xterm** windows, **ae** and **logger** should be false. For other values, use the default.

3. Repeat the **devices** command for each terminal window you want to open. Four windows is a suggested number to get started. When you have completed adding all pty devices, press **F3**.

Devices session ended.

\# _

You are now ready to install the X-Windows licensed program.

4. Locate the X-Windows licensed program diskettes in the X-Windows licensed program diskette binder. You should have three X-Windows licensed program diskettes: two X-Windows program diskettes and one X-Windows samples diskette. Do not put a diskette in the diskette drive until you are prompted to do so.

5. Type `installp`. Then press **Enter**.

```
#  installp
```

You see the following prompt:

```
000-123 Before you continue, you must make sure there is no other
        activity on the system.  You should have just restarted
        the system, and no other terminals should be enabled.  Refer
        to your messages reference book for more information.

        Do you want to continue this command (y or n):
```

See the discussion of message 000-123 in *Messages Reference* if you require more information.

**Warning:** Make sure that you are the only user on your system while you are installing the X-Windows licensed program. Also, you should not be running any programs or have any open files.

The **who** command displays a list of users on the system.

6. To continue, type y. Then press **Enter**.

You see the following prompt:

```
Please mount volume 1 on /dev/rfd0
   . . .   and press Enter to continue
```

7. Insert the first X-Windows licensed program diskette (volume 1 of 2) into the diskette drive, close the diskette drive, and press **Enter**.

You see the following prompt:

```
The program "X-Windows"
will be installed.

Do you want to do this? (y/n)
```

8. To continue the installation, type y and press **Enter**. Copyright information is displayed.

```
IBM RT PC X-Windows Licensed Program
Version 1.1 (C) Copyright International Business Machines Corp. 1987
Licensed Material-Program Property of IBM-All Rights Reserved
RT PC is a trademark of International Business Machines Corp.
Copyright (C) Massachusetts Institute of Technoloty 1985, 1986
Copyright (C) Brown University 1986
Copyright (C) Donald E. Knuth 1985
```

9. The next screen allows you to choose the items to be installed:

```
Choose one or more of the following items to be installed.

1  X-Windows          - base X system and RT PC fonts
2  Fonts              - font tools and other fonts
3  All of the above

To cancel the "installp" command, enter "quit".

To install one or more items, type the ID numbers
separated by spaces (for example:  1 3).  Then press Enter.

---> _
```

To choose an item from the menu, type the ID number for the item you want. Then press **Enter**.

a.  **1 X-Windows** includes the base X-Windows programs as well as the RT PC fonts.

b.  **2 Fonts** includes font tools and additional fonts.

c.  **3 All of the above** includes both items 1 and 2.

10. To proceed with the installation of the X-Windows licensed program, see the pages listed below:

- To install **1 X-Windows**, see "Installing X-Windows" on page A-9.
- To install **2 Fonts**, see "Installing Fonts" on page A-12.
- To install **3 All of the above**, see "Installing All of the X-Windows Programs" on page A-15.

To cancel the **installp** command, type `quit` and press **Enter**.

# Installing X-Windows

Continue with the following procedure if you have chosen X-Windows installation option 1, X-Windows.

1.  After choosing option 1 and pressing **Enter**, you see the following menu:

```
From the list below, choose the language(s) for keyboard mapping.

1   Austrian/German             9   Italian
2   Belgian                    10   Japanese English
3   Canadian ( French)         11   Norwegian
4   Danish                     12   Portuguese
5   English (UK)               13   Spanish
6   English (US)               14   Swiss ( French )
7   Finnish / Swedish          15   Swiss ( German )
8   French (AZERTY)

To cancel the "installp" command, enter "quit".

To install one or more languages, type the group ID numbers
separated by spaces (for example: 1 3).  Then press Enter.
The first number will be the default language used.

---> 6 3_
```

Type the number corresponding to the language you want to use. If you want to use more than one language, type more than one number. Separate numbers with a space. The first number you type is the default language.

2. After you have chosen the language you will use, the following message appears. No action is required at this time.

```
045-001  Installation of "IBM RT PC X Windows" is in progress.
         Installation will take several minutes.
         Time = 01:35
```

3. Insert the first X-Windows licensed program diskette (volume 1 of 2), close the diskette drive door, and press **Enter**. At this time the RT PC loads various X-Windows licensed program files and lists them on the screen.

```
Please mount volume 1 on /dev/rfd0
   . . .  and press Enter to continue
```

4. Insert the second X-Windows licensed program diskette (volume 2 of 2), close the diskette drive door, and press **Enter**. At this time the RT PC loads various X-Windows licensed program files and lists them on the screen.

```
Please mount volume 2 on /dev/rfd0
   . . .  and press Enter to continue
```

5. The following prompts appear; no action is required.

```
045-009  Linking X with GSL.  This will take a few moments.
         Time = 01:35
```

```
045-007  Building character sets for base system:
         Time = 01:35
```

*⟨filename⟩*

6. Installation of X-Windows licensed program is complete when you see the following message.

```
Program "X-Windows"
is now installed.
```

# Installing Fonts

Continue with the following procedure if you have chosen X-Windows licensed program installation option 2, Installing Fonts.

1. After choosing option 2 and pressing **Enter**, you see the following menu. Choose the font group you wish to install.

```
"Fonts" are divided into several groups, each of
which can be separately installed.  The RT fonts are
automatically installed if X-Windows is being installed.
The groups and their ID numbers are:

1  RT Fonts                      - normal, italic, bold, ...
2  VT100 Fonts                   - vtsingle, vtbold, nil2, ...
3  Character Fonts               - 6x10, 8x13, ...
4  Miscellaneous Fonts           - math5, ...
5  Font Tools                    - font compiler, source, ...
6  All of the above

To cancel the "installp" command, enter "quit".

To install one or more groups, type the group ID numbers
separated by spaces (for example: 1 3).  Then press Enter.

---> _
```

2. After you have chosen the font set you will use, the following message appears. No action is required of you at this time.

```
045-001   Installation of "IBM RT PC X Windows" is in progress.
          Installation will take several minutes.

          Time = 01:35
```

3. If not already in the diskette drive, insert the first X-Windows licensed program diskette (volume 1 of 2), close the diskette drive door, and press **Enter**. At this time the RT PC loads various X-Windows licensed program files and lists them on the screen.

```
Please mount volume 1 on /dev/rfd0
 . . .  and press Enter to continue
```

4. Insert the second X-Windows licensed program diskette (volume 2 of 2), close the diskette drive door, and press **Enter**. At this time the RT PC loads various X-Windows licensed program files and lists them on the screen.

   **Note:** This step and the prompt displayed might not occur, depending on which options you selected, and if files are needed from the second diskette.

```
Please mount volume 2 on /dev/rfd0
 . . .  and press Enter to continue
```

5. When you see the next message, installation of X-Windows Fonts is complete.

```
Program "X-Windows"
is now installed.
```

# Installing All of the X-Windows Programs

Continue with the following procedure if you have chosen X-Windows licensed program installation option 3, Installing All of the X-Windows Programs.

1. After choosing option 3 and pressing **Enter**, you see the following menu:

```
From the list below, choose the language(s) for keyboard mapping.

1   Austrian/German          9   Italian
2   Belgian                 10   Japanese English
3   Canadian ( French)      11   Norwegian
4   Danish                  12   Portuguese
5   English (UK)            13   Spanish
6   English (US)            14   Swiss ( French )
7   Finnish / Swedish       15   Swiss ( German )
8   French (AZERTY)

To cancel the "installp" command, enter "quit".

To install one or more languages, type the group ID numbers
separated by spaces (for example: 1 3).  Then press Enter.
The first number will be the default language used.

---> 6 3_
```

Type the number corresponding to the language you want to use.  If you want to use more than one language, type more than one number.  Separate numbers with a space. The first number you type is the default language.

2. After choosing the language you will use, the following is displayed. Choose the font group you wish to install.

```
"Fonts" are divided into several groups, each of
which can be separately installed.  The RT fonts are
automatically installed if X-Windows is being installed.
The groups and their ID numbers are:

1  RT Fonts                           - normal, italic, bold, ...
2  VT100 Fonts                        - vtsingle, vtbold, nil2, ...
3  Character Fonts                    - 6x10, 8x13, ...
4  Miscellaneous Fonts                - math5, ...
5  Font Tools                         - font compiler, source, ...
6  All of the above

To cancel the "installp" command, enter "quit".

To install one or more groups, type the group ID numbers
separated by spaces (for example: 1 3).  Then press Enter.

---> _
```

3. After you have chosen the fonts you will use, the following message appears. No action is required of you at this time.

```
045-001   Installation of "IBM RT PC X Windows" is in progress.
          Installation will take several minutes.
          Time = 01:35
```

4. Insert the first X-Windows licensed program diskette (volume 1 of 2). At this time the RT PC loads various X-Windows licensed program files and lists them on the screen.

```
Please mount volume 1 on /dev/rfd0
 . . .   and press Enter to continue
```

5. Insert the second X-Windows licensed program diskette (volume 2 of 2). At this time the RT PC loads various X-Windows licensed program files and lists them on the screen.

```
Please mount volume 2 on /dev/rfd0
 . . .   and press Enter to continue
```

6. The following messages appear; no action is required.

```
045-009  Linking X with GSL.  This will take a few moments.
         Time = 01:35
```

```
045-007  Building character sets for base system:
         Time = 01:35
```

*⟨filename⟩*

7. Installation of X-Windows licensed program is complete when you see the following message.

```
Program "X-Windows"
is now installed.
```

# Installing Sample X Programs

Continue with the following procedure if you want to install the X-Windows sample programs.

1. Type `installp`. Then press **Enter**.

```
#  installp
```

You see the following prompt:

```
000-123 Before you continue, you must make sure there is no other
        activity on the system.  You should have just restarted
        the system, and no other terminals should be enabled.  Refer
        to your messages reference book for more information.

        Do you want to continue this command (y or n):
```

See the discussion of message 000-123 in *Messages Reference* if you require more information.

2. To continue, type y. Then press **Enter**.

   You see the following prompt:

   ```
   Please mount volume 1 on /dev/rfd0
    . . .   and press Enter to continue
   ```

3. Insert the X-Windows Samples program diskette into the diskette drive, close the diskette drive, and press **Enter**.

   You see the following prompt:

   ```
   The program "X-Windows Samples"
   will be installed.

   Do you want to do this? (y/n)
   ```

4. To continue the installation, type y and press **Enter**. Copyright information is displayed.

```
IBM RT PC X-Windows Samples Program
Version 1.1 (C) Copyright International Business Machines Corp. 1987
Licensed Material-Program Property of IBM-All Rights Reserved
RT PC is a trademark of International Business Machines Corp.
COPYRIGHT (C) Massachusetts Institute of Technology 1985, 1986
COPYRIGHT (C) Digital Equipment Corp., Massachusetts 1985, 1986, 1987
```

5. If not already in the diskette drive, insert the X-Windows Samples diskette. At this time the RT PC loads various X-Windows sample files and lists them on the screen.

```
Please mount volume 1 on /dev/rfd0
 . . .  and press Enter to continue
```

6. When you see the next message, installation of X-Windows Samples is complete.

```
Program "X-Windows Samples"
is now installed.
```

# Installation Requirements for Remote Usage

Before the X-Windows licensed program can be used remotely, certain components in addition to X-Windows must be installed and running on both the host and remote systems. The components required for remote use of X-Windows are:

- The VRM Baseband Adapter Device Driver and IBM RT PC Baseband Adapter

  **OR**

- VRM Token-Ring Device Driver and IBM Token-Ring Network RT PC Adapter

  **OR**

- Both.

TCP/IP is highly recommended for remote login purposes.

Refer to the installation procedures packaged with each licensed program for other installation instructions.

For information on using X-Windows remotely, see "Using X-Windows on a Remote System" on page 2-21.

# Appendix B.  Fonts

This section describes a font support package for the RT PC.  Topics discussed include:

- Font file format used with the X Windows program (**rtx** format)
- Font file naming conventions
- Use of supplied font source files
- Use of a supplied program (METAFONT[1]) to create and modify fonts
- Use of a supplied facility to convert font files to **rtx** format.

A *font* is defined as a complete assortment of any one size and style of type containing all the characters, usually both lowercase and uppercase, alphabetic and numeric.

---

[1]    METAFONT is a trademark of Addison Wesley Publishing Company.

# Overview of Font Support

The font package provided for RT PC applications includes several fonts of various sizes and styles, as well as a program to create and modify fonts, particularly for use with various types of printers. A font that looks acceptable on a display device may be unsuitable for printed output because the resolution in pixels per inch of a printer is different from a display. On a printer, the resulting font size is generally too small to be usable.

METAFONT is a set of font compiler programs that allow you to design or modify your own 256-character fonts. This program, font source files designed for use with the program, and several fonts intended for use with RT PC displays are included with the font package. METAFONT is described in *The METAFONTbook*, by Donald E. Knuth (Addison Wesley Publishing Company, 1986). For details on developing various font styles and sizes with METAFONT, refer to Knuth's *Computer Modern Typefaces* (Addison Wesley Publishing Company, 1986). See "Using METAFONT to Create Fonts" on page B-5 for details on using METAFONT in the X-Windows environment.

Before a font can be used with the RT PC displays, it must be in **rtx** format. In addition to providing a uniform bit storage scheme, the **rtx** format allows for fonts of variable widths and compression techniques. See "Converting Fonts to X-Windows Fonts" on page B-23 for information on converting other font file formats to **rtx** format.

The supplied **rtx** font files, the METAFONT program, and the font source files can be optionally installed from a menu on the X-Windows diskettes. If you choose to install these fonts, they will be stored in **/usr/lpp/fonts**. Additional fonts that you create should also be stored here. The **rtx** font format is described in **/usr/include/sys/font.h**.

If you choose to install the font package, the METAFONT program, font source files for use with METAFONT, and a set of **rtx** format fonts are installed. These fonts have been tuned for legibility on the RT PC displays and include all the characters in code pages P0, P1, and P2. The supplied font source files, from which you can create new fonts with METAFONT, include only the characters from code page P0. The fonts provided with the font package include:

- Rom6.500
- Rom10.500
- Rom14.500
- Rom22.500
- Rom29.500
- Bld14.500
- Itl14.500
- Erg14.500.

For a description of the significance of the fields in the font file names, see "Font File Naming Conventions."

# Font File Naming Conventions

This section describes the naming convention for **rtx** format font files. This convention is intended to associate a meaningful descriptor with a given file and to avoid storing duplicate fonts on the system.

Fonts supplied by IBM or created with METAFONT typically have an alphabetic descriptor of the font style, followed by a numeric value indicating the resulting font point size, separated by a period, on a device of a given density factor. The density factor is determined by multiplying a device's pixels-per-inch density times a scaling factor of 5.

One font supplied with the font package, **Rom10.500**, would be interpreted according to the convention as follows:

**Rom**
A roman font designed specifically for use on the RT PC. The initial capital letter in the font file name indicates that these fonts have been tuned for legibility on RT display devices. Contrast these file names with the untuned font source files listed in "Using METAFONT to Create Fonts" on page B-5. The supplied fonts are divided into four types. They are:

**Rom**    Roman type designed for clarity on RT displays.

**Itl**    Italic Roman type designed for clarity on RT displays.

**Bld**    Boldface Roman type designed for clarity on RT displays.

**Erg**    A sans-serif type designed for clarity on RT displays.

You can develop your own alphabetic descriptors for font styles you create with METAFONT. Note that font files provided with METAFONT and fonts described in Knuth's *Computer Modern Typefaces* use other descriptors, such as **cmr** (for Computer Modern Roman, a specific type of Roman font) and **cmrs** (for Computer Modern Roman slanted, a Roman font with some curve to the characters, but not as severe as italic).

If you are creating many fonts for use with the RT PC, you will probably create your own (or adopt Knuth's) conventions for alphabetic descriptors. Just try to be consistent when you convert the font files to **rtx** format.

**10**
The font is displayed at 10 points (72 points per inch is standard) on a device with the specified density factor (in this case 500).

**. (period)**
Separates the point size from the density factor.

**500**
Determined by multiplying the scale factor five times the density of the device in pixels-per-inch (in this case 100).

Therefore, the font in this example (**Rom10.500**) would produce 10-point roman type on a device that provides 100 pixels per inch. The same font used with a device providing 200 pixels per inch would appear half as large.

The standard of device pixels-per-inch density is established by the IBM 6155 Extended Monochrome Display (APA-16), which has a density of approximately 100 pixels per inch. The density factor for a 6155 display is 500 (5 times 100 pixels per inch). Therefore, on a 6155 display, font file **Rom6.500** produces 6-point RT roman characters; font file **cmr10.500** (an untuned font produced by METAFONT) produces 10-point computer modern roman characters.

Before a font can be used with the RT PC, it must be converted to **rtx** format. At conversion time, you have the opportunity to rename the converted file to the **rtx** conventions. IBM recommends using the **rtx** conventions for consistency among all font files, so you can more easily identify the point size resulting from a given font on a given device, and to avoid duplication of fonts on the system.

# Using METAFONT to Create Fonts

The following section describes commands used to create and modify METAFONT fonts. When installed, the METAFONT program resides in **/usr/lpp/mf**.

The METAFONT port provides several fonts and base files from which additional fonts can be created. For examples of the advanced font styles and characteristics that can be created with this font compiler program, see *Computer Modern Typefaces*. This book contains dozens of unique typefaces created with the METAFONT program, as well as some of the syntax statements that cause the typefaces to be produced.

In addition to the files provided with METAFONT, other font source files, which have already been built with METAFONT, are included.

The font source files represent the characters of code page P0 as implemented by the RT PC. (See *Keyboard Description and Character Reference* for a description of the P0, P1, and P2 characters).

You can copy these files, rename them, and build new characters or font styles into the copies. However, you must be relatively familiar with the techniques described in *The METAFONTbook* and *Computer Modern Typefaces*.

The font source files are installed in **/usr/lpp/fonts** and include the following:

- **rom10p0.mf**
- **rom12p0.mf**
- **rom14p0.mf**
- **bld10p0.mf**
- **itl10p0.mf**

Note that these file names do not begin with a capital letter, indicating that they have not been tuned for the RT displays. The alphabetic descriptors (**rom, itl, bld**) and point sizes are consistent with the naming convention. The P0 stands for code page P0 to differentiate these files from the font files that include characters for P0, P1, and P2. The **mf** suffix identifies these files as METAFONT base files.

To create a font suitable for use on X-Windows from font source file **rom10p0.mf**, perform the following:

1. Issue the **makefont** command.

   ```
   makefont -d rom10p0.mf
   ```

   This creates a font for display only in **pk** format. "makefont" on page B-16 describes the command, its flags, and the METAFONT **pk** format.

2. Issue the **pktortx** conversion command.

   ```
   pktortx rom10p0.590pk rom10p0.500
   ```

This creates an **rtx** format font (untuned, as indicated by the initial lowercase character in the file name) which produces 10-point characters (approximately) on a 100 pixels-per-inch device. The resulting font can be viewed with **gftype**, tuned if necessary, and used with the X-Windows program.

Note that many METAFONT font files actually consist of several **.mf** files. The combined size of the files may be too large to be compiled. For this situation, IBM has included a program to compile a list of files. This program is described in **README.mf** in **/usr/lpp/X/doc**.

3. Issue the **fixrtx** command to ensure that all characters in the font file have any proportional spacing imbedded in the raster image and are a constant height:

```
fixrtx -h rom10p0.500
```

This command converts variable-height characters in an **rtx** font file to constant height characters and builds proportional spacing into a character's raster image. For more information, see "fixrtx" on page B-25.

The METAFONT commands are defined in the following section.

# cmmf

## Purpose

Produces a generic font (**gf**) output file from a METAFONT (**mf**) input file.

## Synopsis

cmmf ── *file* ──┤

## Description

The **cmmf** command invokes the METAFONT program using the Computer Modern **METAFONT** base. This base contains all of the plain base, as well as extensions used by the fonts in the Computer Modern family.

If the *file* argument has no extension, the default extension **.mf** is assumed.

**Note:** The **cmmf** command is similar to the **mf** command ("mf" on page B-20). Use **cmmf** (rather than **mf**) when generating fonts from the Computer Modern source files on **/usr/lpp/mf/macros**.

Output is to one or more files on the current directory. The output files have the same name as the primary input file, but will have one of the following extensions:

*.nnn***gf**     Identifies the generic font (gf) output file, containing the raster data for each character in the file. The value *nnn* is the resolution of the font (in pixels per inch) multiplied by the METAFONT scale factor. For example, the normal resolution of printer fonts is 240 pixels per inch. For a font file called test produced with the **makefont** command with no magnification specified, the output of **cmmf** on test would be test.240gf; at magnification **magstep1** (1.2 times the normal size), the output of **cmmf** on test would be test.288gf. See "makefont" on page B-16 for more details on the **makefont** command.

| | |
|---|---|
| **.tfm** | Identifies a TEX[2] font metric (**tfm**) file. This file is used by the typesetting program TEX when formatting characters from this font. For information on TEX, see *The TEXbook*, by Donald E. Knuth (Addison Wesley Publishing Company, 1986). |
| **.log** | Identifies a file that contains a log of all the terminal input and output (including error messages) that occurs while METAFONT is running. |

The resolution of the generated font depends on the mode setting that METAFONT uses. By default, this mode is proof, which is used for generating proof copies of characters at very high resolution (2602 pixels per inch). To generate fonts for use by actual devices, you must specify either **pageprinter** (for the IBM 3812 Pageprinter or the IBM Quietwriter○ printers), to get 240 pixels-per-inch fonts, **proprinter** (for the IBM 4201 Proprinter) to get 240 pixels-per-inch fonts, or **displays** to get 100 pixels-per-inch fonts. See Example 3 for details on generating fonts for actual devices.

## Examples

1. To run METAFONT on a source file called **cmr10** :

   ```
   cmmf   cmr10
   ```

   The filename extension **.mf** is assumed. Files output by this example would be **cmr10.2602gf** and **cmr10.log**. The resolution of the output is 2602 pixels per inch because the default proof mode is used.

2. To run METAFONT on **cmr10**, but searching for input files from a private macro library using the default shell:

   ```
   MFINPUTS=.:/u/myid/mylib
   export MFINPUTS
   cmmf   cmr10
   ```

   The same example as above, but using the C shell:

   ```
   setenv   MFINPUTS   .:/u/myid/mylib
   cmmf   cmr10
   ```

3. To run METAFONT on source file **cmr10**, to produce a font for the IBM 3812 Pageprinter, specifying **pageprinter** mode instead of the default proof mode:

   ```
   cmmf "\mode=pageprinter; input cmr10"
   ```

---

[2]   TEX is a trademark of the American Mathematical Society.

The output file from this example would be **cmr10.240gf**. The quotes are required to differentiate the ;(semicolon) and the \ (backslash) from the same characters that have special meanings to the shell.

## Files

**/usr/lpp/mf/bases/cm.base**
**/usr/lpp/mf/macros**

## Related Information

In this book: "gftopk" on page B-10, "gftype" on page B-12, "makefont" on page B-16, "mf" on page B-20.

*The METAFONTbook*, by Donald E. Knuth.

*Computer Modern Typefaces*, by Donald E. Knuth.

# gftopk

## Purpose

Converts a generic font (**gf**) file to a packed (**pk**) file.

## Synopsis

gftopk —— *gffilename* —— *pkfilename* ——|

## Description

The **gftopk** command converts a generic font file (*gffilename*) produced with the **cmmf** or **mf** commands to a packed font file (*pkfilename*). Both *gffilename* and *pkfilename* must be specified; there are no default file names or extensions.

The packed font file that is output by **gftopk** is required in order for to convert the METAFONT font to **rtx** format. By convention, the **pk** files produced for the Proprinter have a suffix of **Pk** (note the uppercase P) to distinguish them from the font files for the 3812 and Quietwriter (suffix **pk**). The distinction is necessary because the Proprinter has an aspect ratio of 3:5 rather than 1:1.

## Examples

To convert generic font file **cmr10.240gf** (Computer Modern Roman 10 point for a 240 pixels-per-inch device) to **pk** file **cmr10.1200pk**:

```
gftopk  cmr10.240gf  cmr10.1200pk
```

## Related Information

In this book: "cmmf" on page B-7, "makefont" on page B-16, "mf" on page B-20.

# gftype

## Purpose

Produces an ASCII dump of a METAFONT generic font file.

## Synopsis

gftype ⟨ -m / -p ⟩ file

## Description

The **gftype** command takes a METAFONT **gf** file and produces an ASCII dump of its contents. The resulting dump contains a mnemonic representation of the data for each character, a pixel map of each character drawn as an array of asterisks and blanks, and the **tfm** data. Output is to **stdout** and can be redirected to a file by the normal AIX mechanisms.

## Flags

-m     Deletes the per-character mnemonic information from the dump.

-p     Deletes the per-character pixel maps from the dump.

## Examples

1.  To produce a complete dump (mnemonics, pixel maps, and **tfm** data) of file **cmr10.240gf**:

    ```
    gftype  cmr10.240gf
    ```

2.  To produce a dump containing only the pixel maps and **tfm** data of file **cmr10.240gf**:

    ```
    gftype  cmr10.240gf -m
    ```

## Related Information

In this book: "cmmf" on page B-7, "mf" on page B-20.

*The METAFONTbook*, by Donald E. Knuth.

*The T$_E$Xbook*, by Donald E. Knuth.

# inimf

## Purpose

Produces a METAFONT base file to be loaded by the production version of METAFONT.

## Synopsis

```
inimf
```

See **Examples** for parameter data.

## Description

The **inimf** command initializes the METAFONT program and loads the METAFONT internal tables with macro definitions, parameters, and other initialization values that make up a METAFONT base. The base you get depends on the specification of the command. Possible choices are plain base or Computer Modern base. If you do not specify all the parameters, the system will prompt you for parameters after you enter **inimf**.

The file that is output by **inimf** has an extension of **.base**. From a base file, you can run **cmmf** or **mf** to produce a **gf** font file.

Bases for system-wide use are stored on the METAFONT base library (**/usr/lpp/mf/bases**), but output from **inimf** goes to the current directory. Therefore, any base you create with **inimf** for use by other programs must be moved to the base library.

## Examples

Input to **inimf** can be specified interactively in response to prompts from METAFONT.

1. To create a plain METAFONT base file:

```
inimf "plain; input local; dump"
```

This sequence creates the files **plain.base** and **plain.log** on the current directory. The **local.mf** file is input just before dumping in order to define the printer and display modes to METAFONT. To test your new base, issue the **mf** command. METAFONT searches the current directory first for base files. See "mf" on page B-20 for information on how to use the MFBASES environment variable to control the METAFONT search for base files.

2.  To create the Computer Modern base, **cm.base**:

```
inimf "plain; input cmbase; input local; dump"
```

The **inimf** command creates its output on the file **plain.base**, since **plain** is the first file name it sees. Use the **mv** command to change **plain.base** to the proper name.

## Files

**/usr/lpp/mf/macros**

## Related Information

In this book: "cmmf" on page B-7, "mf" on page B-20.

# makefont

## Purpose

Produces a display font and printer font from one or more METAFONT source files.

## Synopsis

makefont ─┬─ -b ─┬─ file ─┬─ one of ─┬─
          │ -b   │        │ magstephalf │
          │ -d   │        │ magstep1    │
          │ -p   │        │ magstep2    │
          │ -Bnnn│        │ magstep3    │
          │ -P   │        │ magstep4    │
                          │ magstep5    │
                          │ nnn pt      │

-or-

makefont ─┬─ -b ─┬─ list= listfile ─┤
          │ -b   │
          │ -d   │
          │ -p   │
          │ -Bnnn│
          │ -P   │
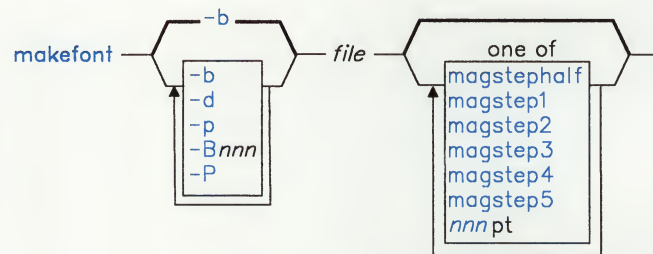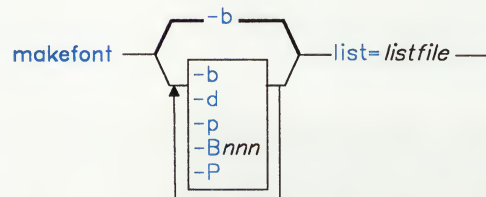
## Description

The **makefont** command generates Computer Modern fonts for use on displays or printers. METAFONT is invoked using the **cm** base. The **makefont** command calls **gftopk** to convert the generic font **gf** file to a packed font (**pk**) file. The input *file*, therefore, should be a Computer Modern source file without the **.gf** extension.

By default, this command generates fonts for both the printer (3812 and Quietwriter) and the displays unless the **-d** or **-p** flags are specified.

The first form of the command generates output from only one font source file. You would use this form to generate a single font of a specific size. For example, suppose you need a font for printed output, but find that the font does not exist on the library in the size that you need. Use **makefont** with the desired magnification to create a font of the required size.

The second form of the command generates output from all the font source files in *listfile*. Each line of *listfile* contains either a comment (first character of the line is **%**) or the operands of the first form of the **makefont** command. For example:

```
% Make a font for a printer only.
makefont cmr10 magstep3 -p
% Make a font for a display only.
makefont cmr10 8pt -d
% Make a font for a printer and display.
makefont cmr12 magstep1 -b
```

If you have write permission to **/usr/lpp/X/font**, you can store newly-created fonts directly to the font library. Otherwise, the fonts you create are stored in the current directory where you are working and must be moved to the font library for system-wide use.

Output to the font library consists of the **tfm** file for the font and one or more packed font (**pk**) files. To avoid accumulating many files when running **makefont** on a long list of font names, METAFONT erases the **.log** file and the **gf** files (when the corresponding **pk** is created).

Output to the current directory includes the **gf**, **pk**, and **.log** files.

METAFONT input files are searched for on the current directory and then on the METAFONT macro library, **/usr/lib/mf/macros**. You can specify your own search order by setting the MFINPUTS environment variable to a list of path names (separated by colons).

The METAFONT base file is searched for on the current directory and then on the METAFONT base library, **/usr/lib/mf/bases**. This search order can be overridden by setting the MFBASES environment variable. If **makefont** determines that a particular font file already exists on the output library (either **/usr/lpp/X/font** or the current directory), it bypasses METAFONT execution for that file. To replace a font file, you must first delete the appropriate **pk** file from the output library. If you find that you have a special font-generation requirement that **makefont** cannot handle, copy **/usr/bin/makefont** to one of your own directories and modify it.

# makefont

## Flags

The first flag can be used to specify whether both printer and display fonts are produced, and can also indicate the base from which to produce the fonts. The second flag can be used to specify the magnification of the resulting font. This is typically used when you have an existing font and simply need to resize it for a printer or a display.

The first set of flags are defined as follows:

**-b**    Generate printer (3812 and Quietwriter only) and display fonts. This is the default.

**-d**    Generate display fonts only.

**-p**    Generate 3812 and Quietwriter fonts only.

**-B***nnn*   Name of the base file to be loaded at the start of METAFONT execution. By default, the **cm** base is used.

**-P**    Generate Proprinter fonts only. To generate all three types of printer fonts, specify the flags **-P -b**, in that order.

The magnification flags are defined as follows:

**magstephalf**    1.095 times the size of the input font file.

**magstep1**    1.2 times the size of the input font file.

**magstep2**    1.44 times the size of the input font file.

**magstep3**    1.728 times the size of the input font file.

**magstep4**    2.074 times the size of the input font file.

**magstep5**    2.488 times the size of the input font file.

**Note:**  The **magstep** flags can be used with any font.

*nnn***pt**    Size is *nnn* points.

This flag works only if *file* ends in a decimal number specifying the design size of the font (such as **cmr10**). The **makefont** command reads this numeric suffix and uses it with the value of *nnn* to create a fractional magnification which it passes to METAFONT. For example, specifying `cmr10 15pt` passes METAFONT a magnification value of 15/10.

## Examples

1. To create both printer and display fonts for the **cmr10** font (normal size):

   ```
   makefont  cmr10
   ```

   This example creates **cmr10.tfm**, **cmr10.590pk** (the display font), and **cmr10.1200pk** (the printer font).

2. To create the same font at magnification **magstep4**:

   ```
   makefont  cmr10 magstep4
   ```

   This example creates **cmr10.tfm**, **cmr10.1223pk** (the display font), and **cmr10.2488pk** (the printer font).

3. To generate a display font only from **testfont.mf**:

   ```
   makefont -d  testfont
   ```

4. To generate display and printer fonts for a font of your own design, using your own **mybase.base** file:

   ```
   makefont  -Bmybase myfont
   ```

## Files

**/usr/bin/makefont**
**/usr/lpp/mf/bases/cm.base**
**/usr/lpp/mf/bases/plain.base**
**/usr/lpp/mf/macros**

## Related Information

In this book: "cmmf" on page B-7, "gftopk" on page B-10, "mf" on page B-20.

*The METAFONTbook*, by Donald E. Knuth.

# mf

## Purpose

Produces a generic font (**gf**) output file from a METAFONT (**mf**) input file.

## Synopsis

mf—— *file* ——|

## Description

The **mf** command invokes the METAFONT program using the plain **METAFONT** base.

If the *file* argument has no extension, the default extension **.mf** is assumed.

**Note:** The **mf** command is very similar to the **cmmf** command ("cmmf" on page B-7). The **mf** command generates fonts from the plain source files on **/usr/lpp/mf/macros**; **cmmf** generates fonts from the Computer Modern source files.

Output is to one or more files on the current directory. The output files have the same name as the primary input file, but will have one of the following extensions:

*.nnn***gf**    Identifies the generic font (gf) output file, containing the raster data for each character in the file. The value *nnn* is the resolution of the font (in pixels per inch) multiplied by the METAFONT scale factor. For example, the normal resolution of printer fonts is 240 pixels per inch. For a font file called test produced with the **makefont** command with no magnification specified, the output of **mf** on test would be test.240gf; at magnification **magstep1** (1.2 times the normal size), the output of **mf** on test would be test.288gf. See "makefont" on page B-16 for more details on the **makefont** command.

**.tfm**    Identifies a T$_E$X font metric (**tfm**) file. This file is used by the typesetting program T$_E$X when formatting characters from this font. For information on T$_E$X, see *The T$_E$Xbook*, by Donald E. Knuth (Addison Wesley Publishing Company, 1986).

**.log**    Identifies a file that contains a log of all the terminal input and output (including error messages) that occurs while METAFONT is running.

The resolution of the generated font depends on the mode setting that METAFONT uses. By default, this mode is proof, which is used for generating proof copies of characters at very high resolution (2602 pixels per inch). To generate fonts for use by actual devices, you must specify a mode of either **pageprinter** to get 240 pixels-per-inch fonts, **proprinter** to get 240 pixels-per-inch fonts, or **displays** to get 100 pixels-per-inch fonts. See Example 3 for details on generating fonts for actual devices.

## Examples

1. To run METAFONT on a source file called **sample.mf**:

```
mf   sample
```

The filename extension **.mf** is assumed. Output is to **sample.2602gf** and **sample.log**. The resolution of the output is 2602 pixels per inch because the default **proof** mode is used.

2. To run METAFONT on **cmr10**, but searching for input files from a private macro library using the default shell:

```
MFINPUTS=.:/u/myid/mylib
export MFINPUTS
cmmf   cmr10
```

The same example as above, but using the C shell:

```
setenv   MFINPUTS   .:/u/myid/mylib
cmmf   cmr10
```

3. To run METAFONT on source file **cmr10**, to produce a font for the IBM 3812 Pageprinter, specifying **pageprinter** mode instead of the default proof mode:

```
cmmf "\mode=pageprinter; input cmr10"
```

The output file from this example would be **cmr10.240gf**. The quotes are required to differentiate the ;(semicolon) and the \ (backslash) from the same characters that have special meanings to the shell.

## Files

/usr/lpp/mf/bases/plain.base
/usr/lpp/mf/macros

## Related Information

In this book: "cmmf" on page B-7, "gftopk" on page B-10, "gftype" on page B-12, "makefont" on page B-16.

*The METAFONTbook*, by Donald E. Knuth.

# Converting Fonts to X-Windows Fonts

Before a font can be used on the RT PC, it must be stored in **rtx** format. All the characters in the **rtx** file must be a constant height for the font to work in the X-Windows environment.

The commands described in the following section can be used to perform the following font-conversion operations:

- Convert an AIX font file to an X-Windows **rtx** font file
- Convert variable-height characters in an **rtx** font file to constant-height characters
- Convert an **onx** font file to an **rtx** font file
- Convert a METAFONT packed font (**pk**) file to an X-Windows **rtx** font file.

# aixtortx

## Purpose

Converts a standard AIX font file to the RT PC **rtx** font file format.

## Synopsis

aixtortx —— *aixfilename* —— *rtxfilename* ——|

## Description

Before any of the standard fonts provided with the AIX Operating System can be used with the X-Windows program, they must be converted to the X-Windows **rtx** font file format. This command takes as input one of the fonts supplied with AIX (from **/etc/vtm**). The output file is in **rtx** format and can be used with RT PC displays.

## Examples

To convert the AIX font file **etc/vtm/itl1.9x20** to an **rtx** font file:

```
aixtortx  /etc/vtm/itl1.9x20  /usr/lpp/fonts/Itl14.500
```

Note how the output file name conforms to the **rtx** naming convention.

## Files

**/etc/vtm**
**/usr/lpp/fonts**

# fixrtx

## Purpose

Converts an **rtx** font file that contains proportional spacing parameters into a file with the proportional spacing built into the raster image.

## Synopsis
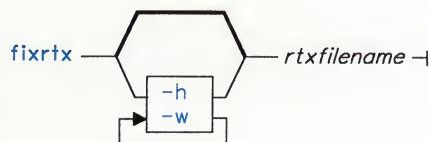


## Description

In order for **rtx** font files to work in the X-Windows environment, any proportional spacing (blank or white space preceding or following the character image) must be contained in the raster image. In addition, all the characters in the font file must be a constant height. (Variable-height characters within a single font file are not allowed.) The **fixrtx** command takes an **rtx** font file and builds any proportional spacing into the raster image. With the **-h** flag of the command, all the characters in the font file are made a constant height. The constant height is determined by the tallest character in the *rtxfilename* file.

This command will not alter the file if it is already in the correct format. You must execute this command (with the **-h** flag specified) on **rtx** files created with the **pktortx** command if they are intended to be used in the X-Windows environment.

The name of the **rtx** font file does not change.

## Flags

-h      Change all characters to a constant height equal to the height of the tallest character in the font. Specify this flag for all font files that are intended to be used in the X-Windows environment.

-w      Change all characters to a constant width equal to the width of the widest character in the font.

## Examples

To convert **rtx** font file **itl10.500** (which includes characters of variable heights) to a file of constant-height characters:

```
fixrtx -h itl10.500
```

The resulting file retains the name of the input file, but the characters are all a constant height and any proportional spacing is built into the character's raster image.

# onxtortx

## Purpose

Converts an **onx** font file to the RT PC **rtx** font file format.

## Synopsis

onxtortx ── *onxfilename* ── *rtxfilename* ─┤

## Description

The **onx** format fonts are typically provided with X-Windows systems provided by the Massachusetts Institute of Technology. Before these fonts can be used with the X-Windows program, they must be converted to the **rtx** font file format. This command takes as input an **onx** font file and produces an **rtx** format file.

## Examples

To convert the **onx** font file **vtsingle.onx** to an **rtx** font file:

onxtortx   vtsingle.onx   /usr/lpp/fonts/vtsingle

## Files

**/usr/lpp/fonts**

# pktortx

## Purpose

Converts a METAFONT **pk** font file to the RT PC **rtx** font file format.

## Synopsis

pktortx —— *pkfilename* —— *rtxfilename* ——⟨ append ⟩——|

## Description

Before any of the standard fonts provided with or produced by the METAFONT program can be used with the X-Windows program, they must be converted to the X-Windows **rtx** font file format. This command takes as input a **pk** METAFONT file (note that this conversion does not work for **gf** type files). The resulting output file is in **rtx** format and can be used with RT PC displays if all the characters in the font file are of a constant height.

Note that **pk** font files typically contain characters of variable widths and variable heights. In this case, the **rtx** font file produced by **pktortx** will also contain characters of variable widths and heights. Therefore, before the **rtx** font file generated by this command can be used in the X-Windows environment, the characters in the file must be converted to a constant height. See "fixrtx" on page B-25 for instructions on this conversion command.

## Flags

The **append** option puts multiple **pk** format files into a single **rtx** file. This option is required if the original source file is too large to be compiled with the **makefont** command. In this case, the source file must be broken down into multiple files. Each file is first compiled into several **pk** files using the **makefont** command. Each **pk** file that was required to contain all the information in the original source file is then appended to *rtxfilename* in **rtx** format.

For example, if a METAFONT source file requires three separate **mf** files in order to compile (and therefore requires three separate **pk** files), the **pktortx** command is first issued to establish the name of the **rtx** file. The command is then issued with the other **pk** file names, the same **rtx** file name, and the **append** option.

```
pktortx /mydir/pkfile1 /mydir/rtxfile
pktortx /mydir/pkfile2 /mydir/rtxfile append
pktortx /mydir/pkfile3 /mydir/rtxfile append
```

This example results in one **rtx** font file, **/mydir/rtxfile**. The user must ensure that each **pk** file to be appended was created using the same parameter files and that each file has unique character codes.

## Examples

To convert the **pk** font file **/usr/lpp/fonts/cmr10.1200pk** to an **rtx** font file:

```
pktortx  /usr/lpp/fonts/cmr10.1200pk  /usr/lpp/fonts/cmr10.1200
fixrtx -h  /usr/lpp/fonts/cmr10.1200
```

Note how the output file name conforms to the **rtx** naming convention (with an expanded alphabetic descriptor classification for Computer Modern Roman).

## Files

**/usr/lpp/fonts**

In this book: "fixrtx" on page B-25

**pktortx**

**access list.** Programs can use the display if they are run on the host system or on any of the systems listed in this file.

**ancestors.** If W is inferior to A, then A is an ancestor of W.

**bitmap.** A pixmap of depth 1.

**button grabbing.** The mouse can be grabbed by a client, either passively by the program itself, or actively by clicking a button.

**child window.** A first-level subwindow.

**client.** An application program connects to X-Windows by some interprocess communication path (IPC) path, such as a TCP connection or a shared memory buffer. The program may be referred to as the client of the server, but it is actually the IPC path itself. Programs with multiple paths open to the server are viewed as multiple clients by the protocol.

**clipping regions.** The image defined by the bitmap or rectangles used to restrict output to a particular region of a window.

**color cell.** An entry in a color map. An entry contains three values: red, green, and blue intensities. The values are 16-bit, unsigned numbers, with 0 representing the minimum intensity. The values are scaled by the server to match the particular display you are using.

**ColorMap.** A set of color cells. A pixel value indexes the color map, producing intensities.

**connection.** The IPC path between the server and a client program.

**coordinate system.** X is horizontal, and Y is vertical. The origin is (0,0) at the upper left.

For a window, the origin is inside the border. Coordinates are discrete and are specified in pixels. Each window and pixmap has its own coordinate system.

**cursor.** The visible shape of the pointer on a screen. In X-Windows, it consists of a hot spot, a source bitmap, and a pair of colors. Defining a cursor for a window controls the appearance of the cursor when it is in that window.

**depth.** The number of bits per pixel used by a window or pixmap.

**device.** In X-Windows, an input device.

**event.** Information generated either asynchronously from a device or as the side-effect of a client request. Events are grouped into types. Events are not sent to a client by the server unless the client has issued a specific request for information of that type. Events are usually reported relative to a window.

**event mask.** The set of event types a client requests relative to a window.

**event synchronization.** Allows synchronous processing of device events. This is helpful when demultiplexing device events to clients results in a conflict. For example, mouse and keyboard events often occur almost simultaneously during window management operations.

**event propagation.** Device-related events propagate from the source window to ancestor windows until some client has expressed interest in handling that type of event, or until the event is discarded explicitly.

**event source.**  The smallest window containing the pointer is the source of a device-related event.

**exposure event.**  Sent to clients to inform them when contents of regions of windows have been lost.  Contents can be lost when windows are obscured or reconfigured.

**font.**  A set of glyphs (usually characters).  The protocol does not translate or interpretate character sets.  The client indicates values used to access the glyph arrays.

**glyph.**  An image, usually of a character, in a font.

**hotspot.**  The spot associated with a cursor corresponding to the coordinates reported for the pointer.

**identifier.**  A unique value associated with a resource that a client program uses to name the resource.  And identifier can be used over any connection to name the resource.

**Inferiors.**  All the subwindows nested below a window.

**input focus.**  Where the main keyboard input goes.  By default, keyboard events are sent to the client using the window the pointer is in.  It is also possible to attach the keyboard input to a specific window.  Events are then sent to the appropriate client regardless of the pointer position.

**input manager.**  A client controlling keyboard input.

**mapping.**  A window on which a map call has been performed.  Unmapped windows cannot be viewed or seen.

**modifier keys.**  Keys such as **Shift**, **Control**, **Alt**, **CapsLock**, and **Shift**.

**multiplex.**  To interleave or simultaneously transmit two or more messages on a single channel.

**padding.**  Bytes inserted in the data stream to maintain alignment of the protocol requests on natural boundaries.

**paint.**  In computer graphics, to shade an area of a display image; for example, with cross-hatching.

**pixel value.**  The number of bit planes used in a particular window or pixmap.  For a window, a pixel value indexes a color map and derives an actual color to be displayed.

**pixmap.**  A three-dimensional array of bits.  A pixmap can be thought of as a two-dimensional array of pixels, with each pixel being a value from ( ) to ($2^\eta$), with N as the depth (z axis) of the pixmap.  A pixmap can also be thought of as a stack of N bitmaps.

**plane mask.**  A bit mask restricting graphics operations to affect a subset of bit planes.  It is stored in a graphics context.

**pointer.**  The device attached to the cursor and tracked on the screen.

**pointing device.**  A device with effective dimensional motion, usually a mouse.  One visible cursor is defined by the core protocol, and it tracks whatever pointing device is attached as the pointer.

**property.**  The name, type, data format, and some data associated with a window.  The protocol does not interpret properties.  They are a general-purpose naming mechanism for clients.  For example, clients can share information such as resize hints, program names, and icon formats with a window manager by using properties.

**property list.**  The list of properties that are defined for a particular window.

**redirecting control.**  A method for preventing particular attempts to change the size or position of a window by transferring the operation to a specified client instead of performing the operation.

**region.** A rectangular area within a bitmap, a pixmap, a screen, or a window.

**reply.** The way information requested by a client program is sent back to the client. Both events and replies are multiplexed on the same connection. Most requests do not generate replies.

**request.** A command to the server. It is a single block of data sent over a connection.

**resource.** Items such as windows, pixmaps, cursors, fonts, and colors. Each has a unique identifier associated with it for naming purposes. The lifetime of a resource is bounded by the lifetime of the connection over which the resource was created.

**root.** In X-Windows, the root of a window is the screen on which the window was created.

**root window.** Covers a screen. It cannot be reconfigured or unmapped, but otherwise performs like any other window.

**rubber-band outline.** A moveable outline.

**server.** Provides the basic windowing mechanism. It handles IPC connections from clients, demultiplexes graphics requests onto screens, and multiplexes input back to clients.

**stipple.** A bitmap used to tile a region. It serves as an additional clip mask for a fill operation with the foreground color.

**tile.** A 16x15 bitmap. Or, the filling of a region with a bitmap.

**unviewable.** A mapped window with an unmapped ancestor.

**viewable.** A mapped window with all of its ancestors also mapped. A viewable window is not necessarily visible.

**window manager.** The client that provides the manipulation of windows on a screen and much of the user interface.

**XYFormat.** The format of a pixmap organized as a set of bitmaps representing individual bit planes.

**ZFormat.** The format of a pixmap organized as a set of pixel values in scanline order.

# Index

## G

## H

## I

## K

## L

## M

## N

unmap window library function   4-106
unviewable   X-3
update mouse library function   4-106
use keymap library function   4-107
using X-Windows on a remote system   2-21

## V

viewable   X-3
volume control library function   4-56

## W

warp mouse   5-32
warp mouse library function   4-107
window event library function   4-108
window manager   X-3

## X

X command   3-27
X Server malloc space   2-20
xclock command   3-30
xhost command   2-23, 3-33
xinit command   3-35
XLookupMapping   3-5
xopen command   3-37
xterm command   2-24, 3-39
xterm datastream support   5-8
XYFormat   X-3

## Z

ZFormat   X-3

**IBM RT PC X-Windows Guide and Reference**

Book Title

SC23-0804

Order No.

## Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y    N    Is the purpose of this book clear?

Y    N    Is the table of contents helpful?

Y    N    Is the index complete?

Y    N    Are the chapter titles and other headings meaningful?

Y    N    Is the information organized appropriately?

Y    N    Is the information accurate?

Y    N    Is the information complete?

Y    N    Is only necessary information included?

Y    N    Does the book refer you to the appropriate places for more information?

Y    N    Are terms defined clearly?

Y    N    Are terms used consistently?

Y    N    Are the abbreviations and acronyms understandable?

Y    N    Are the examples clear?

Y    N    Are examples provided where they are needed?

Y    N    Are the illustrations clear?

Y    N    Is the format of the book (shape, size, color) effective?

### Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

### Optional Information

Your name

Company name

Street address

City, State, ZIP

No postage necessary if mailed in the U.S.A.

Cut or Fold Along Line

Fold and Tape                                                                              Fold and Tape

Austin, Texas 78758-3493
11400 Burnet Rd.
Department 997, Building 998
International Business Machines Corporation

POSTAGE WILL BE PAID BY ADDRESSEE

**BUSINESS REPLY MAIL**

FIRST CLASS     PERMIT NO. 40     ARMONK, NEW YORK

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**IBM**

IBM RT PC Programming
Family

**Reader's Comment Form**

**IBM RT PC X-Windows
Guide and Reference**

SC23-0804

Your comments assist us in improving our products. IBM may
use and distribute any of the information you supply in any way it
believes appropriate without incurring any obligation whatever.
You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation,
program support, and new program literature, contact the
authorized IBM RT PC dealer in your area.

Comments:

Cut or Fold Along Line

Fold and Tape

Austin, Texas 78758-3493
11400 Burnet Rd.
Department 997, Building 998
International Business Machines Corporation

POSTAGE WILL BE PAID BY ADDRESSEE

FIRST CLASS     PERMIT NO. 40     ARMONK, NEW YORK

# BUSINESS REPLY MAIL

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

SC23-0804-00